

# **EMC<sup>®</sup> Documentum<sup>®</sup> Composer**

**Version 6.6**

## **User Guide**

EMC Corporation  
*Corporate Headquarters:*  
Hopkinton, MA 01748-9103  
1-508-435-1000  
[www.EMC.com](http://www.EMC.com)

Copyright© 2008 – 2010 EMC Corporation. All rights reserved.

Published May 2010

EMC believes the information in this publication is accurate as of its publication date. The information is subject to change without notice.

THE INFORMATION IN THIS PUBLICATION IS PROVIDED AS IS. EMC CORPORATION MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WITH RESPECT TO THE INFORMATION IN THIS PUBLICATION, AND SPECIFICALLY DISCLAIMS IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Use, copying, and distribution of any EMC software described in this publication requires an applicable software license.

For the most up-to-date listing of EMC product names, see EMC Corporation Trademarks on [EMC.com](http://EMC.com).

All other trademarks used herein are the property of their respective owners.

# Table of Contents

---

<b>Preface</b>	11
<b>Chapter 1 Documentum Composer</b>	15
Introduction to Composer	15
Headless Composer and Composer UI	15
Installing Composer	16
Installing the lightweight SysObject plug-in	17
Installing other Composer plug-ins	17
Starting Workflow Manager	17
Installing headless Composer	18
UNIX and Linux support in Composer	18
Configuring the connection broker	18
Starting and configuring Composer	19
Configuring the Java JRE and compiler preferences	19
<b>Chapter 2 Managing Projects</b>	21
Composer projects	21
Creating a project	22
Importing a project	23
Composer reference projects	25
Documentum supplied reference projects	25
Designating projects as reference projects	26
Designating reference projects for new Composer projects	26
Designating reference projects for existing Composer projects	27
Composer artifacts	28
Creating an artifact	30
Importing artifacts	31
Configuring project properties	34
Localizing a Composer project	34
Enabling tracing	40
<b>Chapter 3 Converting DocApps and DocApp archives to Composer projects</b>	43
Converting a DocApp to a Composer project	43
Converting a DocApp archive to a Composer project	46
Preparing for DocApp archive conversion	46
Converting a DocApp archive	47
Post-conversion tasks	49
<b>Chapter 4 Composer and the xCelerated Content Platform</b>	51
Tips and considerations for packaging and installing TaskSpace applications or xCP artifacts	52
Composer projects and DAR files	53

Packaging TaskSpace applications .....	54
Packaging a TaskSpace application with Composer.....	54
Packaging a TaskSpace application with headless Composer .....	54
Packaging xCP artifacts.....	56
Packaging xCP artifacts with Composer .....	56
Packaging xCP artifacts with headless Composer.....	56
Installing TaskSpace applications and xCP artifacts.....	58
Installing a TaskSpace application and xCP artifacts with Composer .....	58
Installing TaskSpace applications and xCP artifacts with the DAR Installer .....	58
Installing TaskSpace applications and xCP artifacts with headless Composer .....	59
Building and installing Composer projects that already contain xCP artifacts with headless Composer.....	60
Migrating a TaskSpace application or xCP artifacts from a source environment to a target environment.....	62
Packaging the TaskSpace application or xCP artifacts on the source environment .....	62
Deploying the TaskSpace application or xCP artifacts on the target repository .....	63
Troubleshooting tips .....	63
<b>Chapter 5 Managing Web Services .....</b>	<b>65</b>
Web services .....	65
Configuring DFS module options .....	65
Configuring the DFS services library .....	66
Configuring catalog services.....	67
Viewing Web services .....	69
Filtering services .....	70
Generating a client proxy .....	71
Consuming a service.....	72
Creating a service .....	73
Creating a service from a Java file .....	73
Creating a service from a WSDL .....	75
Modifying catalog and category information .....	75
Publishing a service .....	76
Unpublishing a service .....	77
Exporting a service .....	78
Deploying a service .....	79
<b>Chapter 6 Managing Alias Sets .....</b>	<b>81</b>
Alias, alias values, and alias sets .....	81
Creating an alias set.....	81
Configuring alias values.....	83
<b>Chapter 7 Managing Aspects .....</b>	<b>87</b>
Aspect modules and aspect types .....	87
Creating an aspect type.....	87
Configuring constraint expressions .....	89
Adding aspect attributes .....	90
Configuring the aspect attribute structure .....	90

	Configuring the aspect UI information .....	92
	Adding a tab .....	93
	Creating an aspect module .....	95
	Configuring aspect module deployment .....	97
	Configuring the aspect module runtime environment .....	99
	Configuring the aspect type.....	100
<b>Chapter 8</b>	<b>Managing Formats .....</b>	<b>103</b>
	Formats .....	103
	Creating a format artifact .....	103
<b>Chapter 9</b>	<b>Managing JARs and Java Libraries .....</b>	<b>107</b>
	JAR definitions, JARs and Java libraries .....	107
	Creating a JAR Definition.....	107
	Linking and configuring a Java Library .....	109
<b>Chapter 10</b>	<b>Managing Lifecycles .....</b>	<b>111</b>
	Lifecycles .....	111
	Lifecycle object types .....	112
	Creating a lifecycle .....	112
	Configuring lifecycle properties.....	113
	Adding and configuring lifecycle states.....	115
	Creating a state type .....	118
	Configuring state entry criteria.....	118
	Configuring state actions.....	120
	Configuring repeating attributes.....	120
	Removing repeating attributes values .....	122
	Setting attributes .....	123
	Setting version labels .....	124
	Removing version labels .....	125
	Setting location links.....	125
	Moving all links .....	126
	Removing location links.....	127
	Assigning a document renderer .....	129
	Assigning document owners .....	129
	Setting permission sets.....	130
	Configuring post-change information .....	131
	Configuring state attributes.....	131
	Deleting a state.....	132
	Deleting a lifecycle.....	133
<b>Chapter 11</b>	<b>Managing Methods and Jobs .....</b>	<b>135</b>
	Methods and jobs .....	135
	Creating a method .....	135
	Creating a job.....	137
<b>Chapter 12</b>	<b>Managing Modules .....</b>	<b>141</b>
	Modules.....	141
	Creating a module .....	141
	Configuring module deployment .....	144

	Configuring the module runtime environment .....	145
<b>Chapter 13</b>	<b>Managing Permission Sets (ACLs)</b> .....	149
	Permissions, permission sets, and permission set templates.....	149
	Basic permissions .....	150
	Extended permissions.....	150
	Creating a permission set template .....	151
	Creating a regular or a public permission set.....	153
	Creating an ACL entry owner.....	156
<b>Chapter 14</b>	<b>Managing Procedures</b> .....	157
	Procedures.....	157
	Creating a procedure .....	157
<b>Chapter 15</b>	<b>Managing Relation Types</b> .....	159
	Relation types .....	159
	Creating a relation type.....	159
<b>Chapter 16</b>	<b>Managing Smart Containers</b> .....	163
	Smart containers.....	163
	Constructing a smart container .....	163
	Adding smart container elements .....	165
	Adding a folder .....	165
	Adding a new folder .....	166
	Adding a document .....	167
	Adding a new document.....	168
	Adding a template .....	168
	Adding a placeholder .....	169
	Adding smart container relationships .....	170
<b>Chapter 17</b>	<b>Managing SysObjects</b> .....	171
	SysObjects.....	171
	Creating a SysObject.....	171
	Viewing and modifying SysObject attributes .....	173
<b>Chapter 18</b>	<b>Managing Types</b> .....	175
	Object types .....	175
	Creating a standard object type .....	176
	Attaching aspects .....	179
	Creating a lightweight object type.....	179
	Configuring constraint expressions.....	183
	Adding, deleting, or modifying events.....	184
	Adding type attributes.....	184
	Configuring the attribute structure .....	185
	Configuring attribute constraints.....	186
	Configuring the type attribute UI.....	188
	Configuring conditional attribute values .....	190
	Configuring attribute value mapping.....	191
	Configuring the type UI information .....	192
	Adding a tab .....	193

<b>Chapter 19</b>	<b>Managing XML Applications</b>	195
	Understanding XML applications and the application configuration file.....	195
	Creating an XML Application artifact .....	195
	Viewing or modifying an XML application configuration file .....	196
<b>Chapter 20</b>	<b>Building and Installing a Project</b>	199
	Understanding the build and installation process .....	199
	Configuring the project installation options.....	200
	Adding an owner installation parameter .....	201
	Configuring pre- and post-installation procedures .....	202
	Configuring artifact install options .....	203
	Generating a DAR file.....	205
	Installing a project .....	205
	Creating an installation parameter file .....	208
	Installing a DAR file with the DAR Installer .....	209
<b>Chapter 21</b>	<b>Managing Projects and DAR Files Using Ant tasks and Headless Composer</b>	213
	Creating a headless Composer build .....	213
	Creating Ant scripts to build, modify, and install Composer projects.....	213
	Creating a batch file to setup and run the build.....	215
	emc.importProject .....	216
	emc.createArtifactProject.....	216
	emc.createTaskspaceApplicationProject .....	217
	emc.importArtifacts.....	219
	emc.importContent.....	220
	emc.build.....	220
	emc.dar.....	221
	emc.install.....	221
	Installing a DAR file with headless Composer on UNIX and Linux systems.....	222
<b>Chapter 22</b>	<b>Working with Source Control Systems</b>	225
	Using a source control system .....	225
	Checking in projects .....	225
	Checking out and importing projects .....	226
	Building the project .....	227
<b>Glossary</b>		229

## List of Figures

Figure 1.	Composer project folders .....	23
Figure 2.	Project properties .....	34
Figure 3.	Structure section in Aspect Attributes view .....	91
Figure 4.	Aspect UI information view .....	92
Figure 5.	Lifecycle properties tab .....	114
Figure 6.	Lifecycle editor with state diagram .....	116
Figure 7.	Lifecycle state actions.....	120
Figure 8.	Structure section in Type Attributes view .....	185
Figure 9.	Attribute constraints .....	187
Figure 10.	Type attribute UI view .....	188
Figure 11.	Conditional assistance view .....	190
Figure 12.	Value mapping table .....	191
Figure 13.	Type UI information view .....	192



# List of Tables

Table 1.	UI-based and Headless Composer Comparison .....	16
Table 2.	Documentum artifacts .....	28
Table 3.	Project and repository information .....	32
Table 4.	Repository information .....	45
Table 5.	DocApp archive import properties .....	48
Table 6.	Migration repository information .....	48
Table 7.	Service registry options.....	68
Table 8.	Web service information .....	76
Table 9.	Publish service information.....	77
Table 10.	Export service information .....	78
Table 11.	Alias details .....	83
Table 12.	Alias type values .....	84
Table 13.	Aspect information on General tab .....	88
Table 14.	Attribute structure properties.....	91
Table 15.	Aspect UI information .....	93
Table 16.	Tab configuration properties.....	94
Table 17.	Properties in module editor General tab.....	96
Table 18.	Module runtime environment properties .....	99
Table 19.	Aspect type properties .....	101
Table 20.	Format artifact properties.....	104
Table 21.	JAR definition properties .....	108
Table 22.	Java library properties.....	110
Table 23.	Lifecycle properties tab parameters.....	114
Table 24.	State properties in Overview tab.....	116
Table 25.	State entry criteria .....	119
Table 26.	Add repeating attribute properties.....	121
Table 27.	Remove repeating attribute properties .....	123
Table 28.	Set attribute properties.....	124
Table 29.	Location link properties .....	126
Table 30.	Move all links properties.....	127
Table 31.	Remove location link properties .....	128
Table 32.	Document owner properties.....	130
Table 33.	Permission set properties .....	131
Table 34.	Method artifact properties.....	136
Table 35.	Job properties.....	138
Table 36.	Properties in General tab.....	142
Table 37.	Module runtime environment properties .....	146

Table 38.	Basic permissions .....	150
Table 39.	Extended permissions.....	150
Table 40.	ACL entry details – Permission Set Template .....	153
Table 41.	ACL entry details – Permission Set .....	155
Table 42.	Relation type properties .....	160
Table 43.	Smart container properties .....	164
Table 44.	Folder properties .....	166
Table 45.	New folder properties.....	166
Table 46.	Document instance properties .....	167
Table 47.	New document instance properties.....	168
Table 48.	Template properties .....	169
Table 49.	Placeholder properties .....	170
Table 50.	SysObject properties .....	172
Table 51.	Type information on General tab .....	176
Table 52.	Lightweight type information on General tab .....	180
Table 53.	Attribute structure properties.....	186
Table 54.	Attribute constraint properties .....	187
Table 55.	Type attribute UI properties .....	188
Table 56.	Input mask examples .....	189
Table 57.	Conditional value properties .....	190
Table 58.	Type UI information .....	192
Table 59.	Tab configuration properties.....	194
Table 60.	Project installation options .....	201
Table 61.	Owner installation parameters .....	202
Table 62.	Artifact installation options .....	204
Table 63.	Install parameter information .....	206
Table 64.	DAR Installer fields .....	210
Table 65.	emc.importProject task.....	216
Table 66.	emc.createArtifactProject task.....	216
Table 67.	projectReferences element .....	217
Table 68.	project element.....	217
Table 69.	emc.createArtifactProject task.....	218
Table 70.	projectReferences element .....	218
Table 71.	project element.....	218
Table 72.	emc.importArtifacts task .....	219
Table 73.	objectIdentities element.....	219
Table 74.	emc.importContent task.....	220
Table 75.	emc.build task .....	220
Table 76.	emc.dar task.....	221
Table 77.	emc.install task.....	222

# Preface

---

This guide describes how to use Documentum Composer to develop enterprise applications and deploy these applications on Documentum Content Server.

## Intended audience

This guide is for users who are developing applications for Documentum Content Server. This guide assumes that the user has a basic understanding of the Documentum platform and content management in general.

## Typographic conventions

The following table describes the typographic conventions used in this guide.

### Typographic conventions

Typeface	Text type
Body normal	In running text: <ul style="list-style-type: none"><li>• Interface elements (button names, dialog boxes)</li><li>• Java classes, interface names</li><li>• Names of resources, attributes, pools, Boolean expressions, buttons, DQL statements, keywords, and clauses, environment variables, functions, menus, utilities</li><li>• Pathnames, URLs, filenames, directory names, computer names, links, groups, service keys, file systems, environment variables (command line and text), notifications</li></ul>
Body normal double quotes	Chapter and section titles

Typeface	Text type
<b>Body Bold</b>	In procedures: <ul style="list-style-type: none"><li>• User actions (what the user clicks, presses, selects, or types) in procedures</li><li>• Interface elements (button names, dialog boxes)</li><li>• Key names</li></ul> In running text: <ul style="list-style-type: none"><li>• Command names, daemons, options, programs, processes, notifications, system calls, man pages, services, applications, utilities, kernels</li></ul>
<i>Body Italic</i>	<ul style="list-style-type: none"><li>• Book titles, emphasis (glossary terms, See also index references)</li><li>• Variables in text (outside of command sample)</li></ul>
Courier	In procedures: <ul style="list-style-type: none"><li>• If shown on separate line, prompts, system output, filenames, pathnames, URLs, syntax examples</li></ul>
<b>Courier Bold</b>	<ul style="list-style-type: none"><li>• User input shown on separate line</li></ul>
<i>Courier Italic</i>	In procedures: <ul style="list-style-type: none"><li>• Variables in command strings</li><li>• User input variables</li></ul>

## Support information

Documentum's technical support services are designed to make your deployment and management of Documentum products as effective as possible. The *Customer Guide to EMC Software Support Services* provides a thorough explanation of Documentum's support services and policies. You can download this document from the Powerlink website (<http://powerlink.EMC.com>).

## Revision history

The following changes have been made to this document.

**Revision history**

Revision date	Description
May 2010	Initial Publication



## Documentum Composer

This chapter contains the following topics:

- [Introduction to Composer, page 15](#)
- [Installing Composer, page 16](#)
- [UNIX and Linux support in Composer, page 18](#)
- [Configuring the connection broker, page 18](#)
- [Starting and configuring Composer, page 19](#)
- [Configuring the Java JRE and compiler preferences, page 19](#)

## Introduction to Composer

Documentum Composer provides tools to create and customize applications for Documentum Content Server. These applications specify how Content Server handles different types of content.

Composer is an Eclipse-based product, a stand-alone program built with the Eclipse platform. Since Composer is a stand-alone program, it contains all the required code and plug-ins. Composer is delivered in the form of a compressed .zip file that is extracted to a directory on the local development machine.

## Headless Composer and Composer UI

There are two Composer versions, Composer and headless Composer. Composer is the full IDE that provides a user interface for creating, building, and installing Composer projects. Headless Composer is a command-line driven build tool for creating, building, and installing Composer projects with Ant tasks. The Ant tasks let you integrate the building of Documentum projects and installing of DAR files into standard Ant build scripts. Because the Ant tasks leverage Composer and Eclipse infrastructure, any build scripts that use these tasks must be executed through the Eclipse AntRunner. For more information on Ant, see <http://ant.apache.org>. The following table describes the differences between the two Composer packages.

**Table 1. UI-based and Headless Composer Comparison**

Features/Functionality	UI-based Composer	Headless Composer
Create new project	Yes	Yes
Create new artifacts	Yes	No
Import DocApps from repository	Yes	No
Import DocApp archives	Yes	No
Import project from local directory	Yes	Yes
Import artifact from repository	Yes	Yes
Build project	Yes	Yes
Install project	Yes	Yes
Install DAR file	No	Yes
	<p>The Composer UI lets you install the project, a process that automatically generates and installs a DAR file “behind the scenes.” However, there is no separate “Install DAR File” option in the Composer UI. If you want to install a DAR file interactively, use the DAR Installer.</p>	<p>Headless Composer lets you install a DAR file using the <b>emc.install</b> Ant task.</p>

## Installing Composer

Documentum Composer is packaged as a compressed zip file that contains the Eclipse platform and all required plug-ins. Installing Documentum Composer involves unzipping the zip file to a directory of your choice.

Before installing Composer, ensure that you meet the following prerequisites:

- Documentum 5.3 SP6 or later repositories
- Java JDK 1.5

### To install Composer:

1. Extract the content of the DCTM\_Composer\_<version>.zip file to a directory on your local machine. A directory named Composer is created.
2. Set the JAVA\_HOME environment variable on your local machine to point to your installation of Java JDK 1.5. For example, if the Java JDK is installed in the C:\Program Files\Java\jdk1.5.0\_17 directory, set the JAVA\_HOME variable to that path.



3. Edit the <Composer\_root>\plugins\com.emc.ide.external.dfc\_1.0.0\documentum.config\dfc.properties file and add the connection broker information, like the following:

```
dfc.docbroker.host[0]=[Repository IP address or host name]
```

If you want to work with lightweight SysObjects, install the lightweight SysObject plug-in as described in [Installing the lightweight SysObject plug-in, page 17](#).

## Installing the lightweight SysObject plug-in

The lightweight SysObject plug-in is not part of the main Composer distribution and must be installed in the <Composer\_root>/plugins directory after you have installed Composer.

### To install the lightweight SysObject plug-in:

1. Download the LightweightObject\_<version>.zip file from the download site (<http://powerlink.EMC.com>).
2. Extract the plug-in to the same directory as Composer. For example, if you extracted Composer to the C:\ directory, extract the LightweightObject\_<version>.zip file to the C:\ directory.
3. Change to the <Composer\_root>/plugins directory and verify that the com.emc.ide.artifact.lwdclass\_1.0.0.jar and com.emc.ide.artifact.lwdclass\_ui\_1.0.0.jar files are in the directory.

## Installing other Composer plug-ins

Composer plug-ins that offer additional functionality and are not part of the main Composer distribution must be installed in the **../Composer/plugins** directory after you have installed Composer.

Depending on how the plug-ins are packaged, you might need to extract the package to the main Composer directory on your local machine or extract the package to a temporary directory and then copy the plug-in file to the **../Composer/plugins** directory.

## Starting Workflow Manager

Workflow Manager is only bundled but not integrated with Composer. When you install Composer, Workflow Manager is extracted to the **../Composer/WorkflowManager** directory on your machine.

### To start Workflow Manager:

1. Change to the **../Composer/WorkflowManager** directory on your machine.
2. Double-click **launch\_wfm.bat**.

The Workflow Manager editor and login dialog display. For more information about Workflow Manager, see the *Workflow Manager User Guide*.

# Installing headless Composer

Headless Composer is distributed in a different .zip file than the UI-based Composer package. The headless Composer .zip file has the format **DCTM\_Headless\_Composer\_<platform>\_<version>.zip**.

## To install headless Composer:

1. Extract the headless Composer zip file to a directory on your local machine. The directory name must not contain any spaces. The headless Composer zip file has the format `DCTM_Headless_Composer_<platform>_<version>.zip`.
2. Edit the `<Composer_root>/plugins/com.emc.ide.external.dfc_1.0.0/documentum.config/dfc.properties` file and add the connection broker information, like the following:  

```
dfc.docbroker.host[0]=[Repository IP address or host name]
```

You must have a valid username and password for all of the repositories that you want to access and that the connection broker is aware of these repositories.

To use headless Composer, see the [Chapter 21, Managing Projects and DAR Files Using Ant tasks and Headless Composer](#)

# UNIX and Linux support in Composer

You can use headless Composer on UNIX and Linux systems to install DAR files to Content Server repositories on UNIX, Linux, and Windows systems. Only the headless Composer distribution that is bundled with Content Server is supported in UNIX and Linux environments.

Alternatively, you can use the DAR Installer or headless Composer on Windows systems to install DAR files to Content Server repositories on UNIX and Linux systems.

See [Chapter 21, Managing Projects and DAR Files Using Ant tasks and Headless Composer](#) for information on how to run headless Composer with Ant tasks.

# Configuring the connection broker

Every time you want to import or import a project or artifacts, you access a Documentum repository. The Documentum connection broker handles repository access. You can update the connection broker at any time.

## To configure the connection broker:

1. Edit the `<Composer_root>/plugins/com.emc.ide.external.dfc_1.0.0/documentum.config/dfc.properties` file and add the connection broker information, like the following:  

```
dfc.docbroker.host[0]=[Repository DocBroker IP address or host name]
```
2. Save your changes.

# Starting and configuring Composer

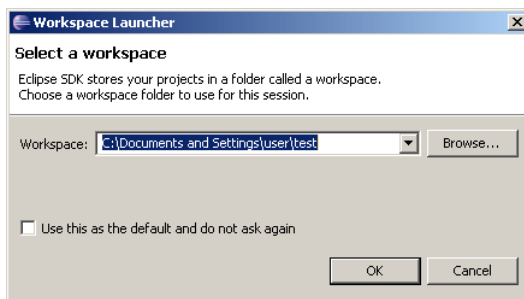
Composer runs on top of the Eclipse platform and uses a similar development concept. In order to run Composer, configure at least one workspace. The workspace is the directory where Composer stores your work. Specify the location for the workspace before you can use Composer.

**Note:** Newer versions of Composer cannot use workspaces created by an older version of Composer. You must create a workspace first and then import projects from the old workspace into the new workspace.

## To start Composer and configure a workspace:

1. Go to the **..\Composer** installation directory on the machine where you extracted the Composer .zip file and double-click the **eclipse.exe** icon.

When you start Composer for the first time, you see the **Workspace Launcher** dialog that allows you to select the location of your workspace.



The workspace is where Composer stores all the source files and dependencies for your projects. You can have more than one workspace in Composer, for example for different projects, but an individual project can only be stored in one workspace.

2. Accept the default location for your workspace or enter a new location in the **Workspace** field, then click **OK**.

The Composer workbench appears.

# Configuring the Java JRE and compiler preferences

The installed Java Runtime Environment (JRE) in the Composer preferences must match the Java Development Environment (JDK) that is configured in the environment variables on the local machine that is running Composer. If the JRE does not match, the Composer project might not install correctly in a repository.

**Note:** Composer requires JRE 1.5. If your local machine has an earlier JRE version installed, upgrade Java before you proceed.

**To configure the Java JRE and compiler preferences:**

1. In a command prompt window, enter SET JAVA\_HOME to verify the path that is set in the JAVA\_HOME environment variable. The JAVA\_HOME variable must point to a JDK 1.5 directory. Your JDK 1.5 directory also contains a JRE directory that you must tell Composer to use as its runtime.
2. In the Composer main menu, select **Window > Preferences**.  
The **Preferences** dialog appears.
3. Click the **Java** option to expand it, then click **Installed JREs**. The **Installed JREs** page appears.
4. If the default installed JRE is not the one that is bundled with your JDK, click **Add** to add another JRE.  
The **Add JRE** dialog appears.
5. Click **Browse** and select the JRE 1.5 directory that came bundled with JDK 1.5, for example C:/Program Files/Java/jdk1.5.0\_17/jre.
6. Click **OK** to verify that the new JRE is on the Installed JREs page and ensure that it is checked.
7. Select **Java > Compiler** from the tree on the left and set the **Compiler compliance level** to 1.5.
8. Click **OK** to save your changes.


# Managing Projects

This chapter contains the following topics:

- [Composer projects, page 21](#)
- [Composer reference projects, page 25](#)
- [Composer artifacts, page 28](#)
- [Configuring project properties, page 34](#)
- [Localizing a Composer project, page 34](#)
- [Enabling tracing, page 40](#)

## Composer projects

A Composer project specifies the objects that make up an application. Therefore, you need to create a project before you can start developing a new application.

A project consists of a project folder and a number of subfolders that contain the artifacts, such as lifecycles, permission sets, jobs, and others. For a complete list of artifacts, see [Table 2, page 28](#). A Composer project is marked with an  icon.

There are several ways to create a Composer project:

- Create a new, empty project as described in [Creating a project, page 22](#).
- Import an existing project into Composer as described in [Importing a project, page 23](#).
- Create a Composer project from a local 5.3 DocApp archive as described in [Converting a DocApp archive, page 47](#).
- Create a Composer project from a 5.3 DocApp, as described in [Converting a DocApp to a Composer project, page 43](#).

**Note:** Newer versions of Composer cannot use workspaces created by an older version of Composer. You must create a workspace first and then import projects from the old workspace into the new workspace.

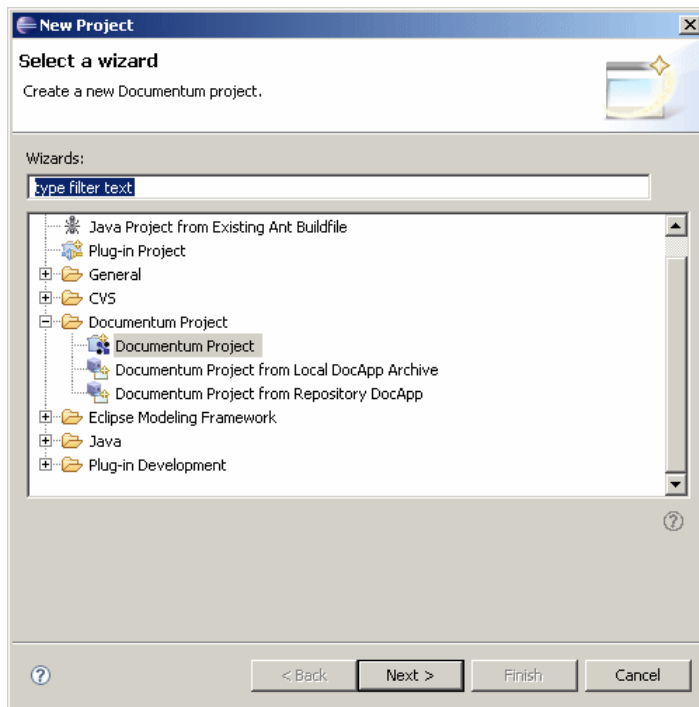
## Creating a project

Create new project whenever you want to create an application from scratch.

### To create a project:

1. Open the **New Project** wizard in one of the following ways:
  - From the main menu in Composer select **File > New > Project**.
  - Right-click in the Documentum Explorer space and select **New > Project**.

The **New Project** dialog appears.



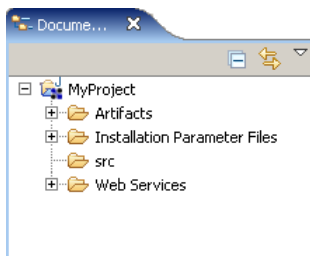
2. Double-click the **Documentum Project** folder to expand it, then select **Documentum Project** and click **Next**.

The **New Documentum Project** dialog appears.

3. Enter a name for your project in the **Name** field and an optional description and click **Next**.
4. Select any projects that you want to designate as reference projects and click **Finish**. For more information about reference projects, see [Composer reference projects, page 25](#)

**Note:** When you create a project for the first time, a dialog box prompts you to select the associated **Documentum Development** perspective. Click **Yes** to select the **Documentum Development** perspective.

Composer creates the project in the **Documentum Navigator** view ([Figure 1, page 23](#)).

**Figure 1. Composer project folders**

By default, a project contains an **Artifacts** folder, **Installation Parameter Files** folder, **src** folder, and a **Web Services** folder, as follows:

- **Artifacts**

The **Artifacts** folder contains subfolders for all artifacts that are available in Composer. When you create a project, these artifact subfolders are empty.

- **Installation Parameter Files**

The **Installation Parameter Files** folder is used for storing the installation parameter files for installing a project. By default, this folder is empty when you create a project. Once you have added artifacts and configured installation options for the project and artifacts, the associated **.installparam** installation parameter files are stored in this folder.

- **src**

The **src** folder is used for storing source files that you want to add to your project. By default, the **src** folder is empty when you create a project.

- **Web Services**

The **Web Services** folder contains Web services files, such as client libraries, WSDL files, and source code files. By default the **Web Services** folder is empty when you create a project.

## Importing a project

You can import existing projects from a local directory into the Composer workspace. If you are using a source control system to manage your files, checkout the project from the source control system before importing it into Composer.

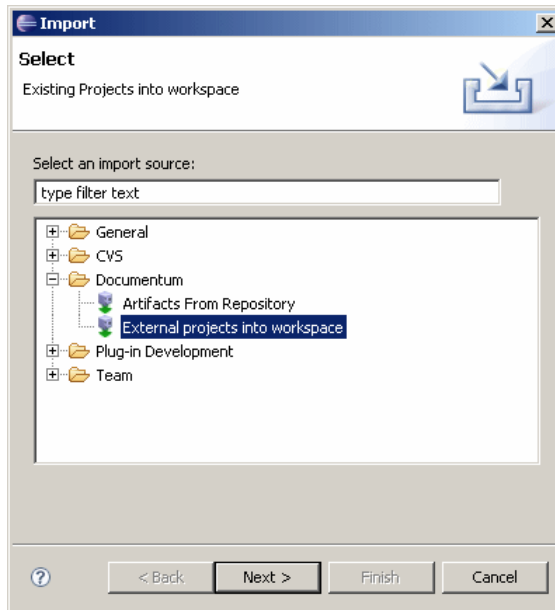
This section describes how to import projects from a local directory, for more information about using Composer with a source control system, see [Chapter 22, Working with Source Control Systems](#).

**Note:** You cannot import a DAR file into a project. A DAR file is the executable version of a project that gets installed in a Documentum repository. A DAR file contains only the binary files of a project but not the source files.

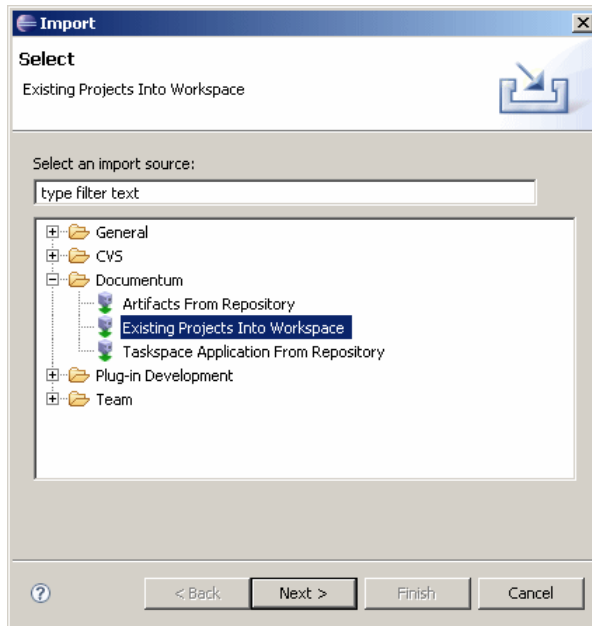
### To import an existing project:

1. Start the **Import** wizard in one of the following ways:
  - Right-click the workspace and select **Import** from the pop-up menu.
  - In the main Composer menu, click **File > Import**.

The **Import** wizard appears.



2. Expand the **Documentum** folder, select **Existing projects into workspace**, then click **Next**. The **Import Projects** dialog appears.



3. Do one of the following:
  - Click the **Select root directory** radio button. Enter the project directory or click **Browse** to search for the directory.
  - Click the **Select archive file** radio button. Enter the archive file or click **Browse** to search for the file.



**Note:** The archive file must have the format .jar, .zip, .tar, .tar.gz, or .tgz. You cannot import Documentum archive (DAR) files. DAR files contain only the binary files of a project but not the source files.

Composer displays the available projects in the **Projects** list box.

4. Select one or more projects to import and select **Copy projects into workspace**, then click **Finish** to import the projects.

Composer imports the projects and displays them in the **Documentum Navigator** view.

**Note:** Composer does not support importing renditions of documents.


## Composer reference projects

Composer lets you create references between projects. This is useful if you have projects that share resources such as Documentum artifacts, libraries, or JAR files. You can specify reference projects when you create a project or by editing an existing project.

In general, you can designate any project as a reference project if it has resources that you want to share with other projects. Documentum also supplies special reference projects that allow you to access Documentum functionality.

## Documentum supplied reference projects

Documentum supplied reference projects are non-buildable projects that you need if you want to use or extend Documentum artifacts (more specifically, Documentum artifacts with names that begin with 'dm').

Every project that you create within Composer has the DocumentumCoreProject designated as a reference project by default. The DocumentumCoreProject contains all of the artifacts that Content Server provides, so you can use or extend these artifacts out of the box. The project is read only and should not be modified. It is marked with the  icon and is displayed only in the **Package Explorer** view, and not the **Documentum Navigator** view. In addition to DocumentumCoreProject, the TCMReferenceProject is also automatically assigned as a reference project if your project contains any TCM artifacts, such as TaskSpace types.

If you use or extend an artifact from another Documentum product, obtain the reference project that contains the artifacts that you want to use. The various Documentum products supply their Composer reference projects in their respective download areas on the EMC software download site, <https://emc.subscribenet.com>.

It is useful to know the following points, which help you understand when to download and reference a Documentum supplied reference project:

- Your Composer project cannot contain artifacts with names that begin with 'dm'. 'Dm' is a reserved prefix for Documentum. Because 'dm' is a reserved prefix, 'dm' artifacts that are present in user projects are detected as errors by Composer. A 'dm' artifact, however,

can exist in Documentum supplied reference projects, such as DocumentumCoreProject or TCMReferenceProject. This provides you with a mechanism to use and extend 'dm' artifacts.

- You can use or extend any 'dm' artifact that DocumentumCoreProject or TCMReferenceProject provides without needing to download a separate reference project. Obtain the reference projects for all other 'dm' artifacts that you require.
- If you import an artifact from the repository, it might depend on other artifacts to function. If these other artifacts are not present in your project or in reference projects, Composer automatically imports these artifacts from the repository. If the automatically imported artifacts have names that begin with 'dm,' it causes the following error:

"Type name is invalid. Type names must not begin with 'dm'. For more information, see the 'Reference projects' section in the Composer User Guide."

If this error occurs, delete the newly imported artifacts, import and designate the appropriate projects as reference projects, and re-import the desired artifacts.

- If you import an artifact that indirectly references a 'dm' artifact, import the project that contains the 'dm' artifact and designate it as a reference project. For example, if you are importing a type named my\_child\_type that depends on a type named my\_parent\_type that depends on a 'dm' type, then download the project that contains the 'dm' type, import it into your workspace, and designate it as a reference project.
- The previous points also apply to converting DocApps and DocApp archives as well. If the DocApp or DocApp archive uses or extends 'dm' artifacts that are not in DocumentumCoreProject, import all required Documentum supplied reference projects into your workspace before converting the DocApp or DocApp archive. During the conversion, Composer prompts you to specify the necessary reference projects.

## Designating projects as reference projects

There are two ways to designate projects as reference projects:

- During the creation of a new project
- By editing an existing project

If you are converting a DocApp or DocApp archive, designate reference projects during the creation of the project.

If you are importing an artifact from the repository that requires a Documentum supplied reference project, designate the reference project first, before importing the artifact. You can do this when creating a project or by editing an existing project.

## Designating reference projects for new Composer projects

If you know beforehand that your project uses or extend 'dm' artifacts that are not in DocumentumCoreProject or TCMReferenceProject, obtain the appropriate reference projects and import them into your workspace. When the import is completed, you can designate the projects as reference projects.

Follow this procedure if you are converting DocApps or DocApp archives into Composer projects and those DocApps or DocApp archives use or extend a 'dm' artifact that is not in DocumentumCoreProject or TCMReferenceProject.

If you are importing 'dm' artifacts or artifacts that extend a 'dm' artifact that is not in DocumentumCoreProject or TCMReferenceProject, follow this procedure as well.

### To obtain and import Documentum supplied reference projects into your workspace:

1. Go to the EMC download site, <https://emc.subscribenet.com/>, to download the necessary reference projects. The reference projects are located in the Documentum product's download area.
2. Import the .zip file of the reference project into your Composer workspace as described in [Importing a project, page 23](#). When the import is complete, the project appears in the Documentum Navigator view of Composer.
3. Create a project from scratch or from a DocApp or DocApp archive as described in the following sections. When prompted, designate the appropriate reference projects.
  - To create a project from scratch, see [Creating a project, page 22](#).
  - To create a Composer project from a 5.3 DocApp. see [Converting a DocApp to a Composer project, page 43](#).
  - To create a Composer project from a local 5.3 DocApp archive, see [Converting a DocApp archive, page 47](#).

## Designating reference projects for existing Composer projects

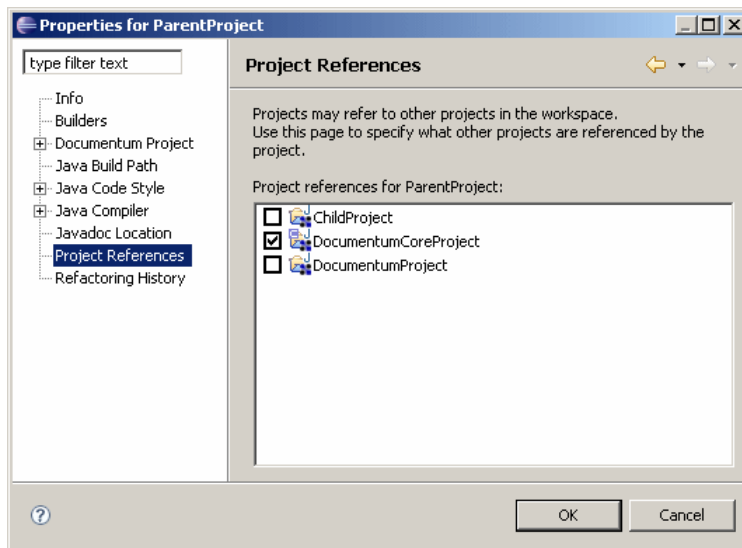
This procedure describes how to specify reference projects for an existing project. You can also specify reference projects when creating a project with the New Project wizard.

Before you can designate a project as a reference project, the project must be in your Composer workspace. If it is not, import the project first, as described in [Importing a project, page 23](#).

**Note:** If you created a project from a DocApp and want to reference Documentum supplied reference projects, do not follow this procedure. Import the Documentum supplied reference projects into your workspace first and select them when prompted by the New Project wizard. If you do not, you will encounter errors during the import process.

### To create a reference to another project:

1. In the **Documentum Navigator** view, right-click the project for which you want to create a reference and select **Properties** from the drop-down list.  
The **Properties** dialog appears.



2. Select **Project References**. The projects that are available for referencing are displayed in the **Project references for ParentProject** list. Select one or more projects that are referenced by this project.
3. Click **OK**.

**Note:** When you are ready to install your project and the project references other projects, be sure to install all projects in the correct order. For example, if project B references artifacts in project A, then project A must be installed first.

## Composer artifacts

Artifacts are Documentum resources, for example object types, modules, and procedures. In Composer, you can create new artifacts or add artifacts to an application using different kinds of wizards and dialogs.

Documentum Composer offers various artifacts, as described in [Table 2, page 28](#).

**Table 2. Documentum artifacts**

Artifact name	Description
Alias Set	Collection of aliases. An alias is a symbolic name that is used as a reference to an actual user, group, cabinet, or folder name. A collection of aliases is called an alias set.
Aspect Module	Customizes behavior for an instance of an object type.
Aspect Type	Specifies the metadata for an instance of an object type.
BAM Configuration	Configuration file used to drive BAM.
BAM Report	Report created using Business Activity Monitor and imported into Composer so that it can be deployed to another repository. BAM Reports can be used to track key performance issues such as SLA enforcement, cycle time, and transaction revenue.

Artifact name	Description
BPM	Process Template created in Process Builder. Can be imported into Composer so that it can be deployed to another repository.
Folder	Folder object within a Documentum repository that is used as a containment structure. A folder contains Documentum objects such as documents.
Group	Defines a logical grouping of users or child groups.
Format	Contains information about a file format recognized by Content Server. A predefined set of file formats is installed by default when a repository is configured.
Form Template	Identifies a functional element for use within a DocApp. You cannot create a form template by using Composer. However you can import forms from an existing DocApp.
Installation Parameter	Specifies installation options that apply to the entire project, such as pre- and post-installation procedures and upgrade options.
Jar Definition	Encapsulates a JAR file. A Java ARchive or JAR file is an archive file that aggregates many files into one.
Java Library	Encapsulates a Java library. A Java library contains interface JARs and implementation JARs that can be linked to other artifacts, such as modules.
Job	Automates the execution of a method, for example how to transfer content from one storage place to another. The attributes of a job define the execution schedule and turn execution on or off.
Lifecycle	Specifies business rules for changes in the properties of an object, such as a document, as it moves through different stages during a business process.
Method	Executable program that is represented by method objects in the repository.
Module	Units of executable code.
Permission Set	Configurations of basic and extended permissions assigned to objects in the repository that lists users and user groups and the actions they can perform.
Procedure	A Docbasic or Java program. Procedures are typically used to specify pre- and post-installation tasks.
Relation Type	Defines the relationship between two objects in a repository.
SysObject	The parent type of the most commonly used objects in the Documentum system. The SysObject type has properties that it passes on to all its subtypes.
Type	Represents a class of objects. The definition of an object type is a set of attributes. The attribute values describe individual objects of the type.

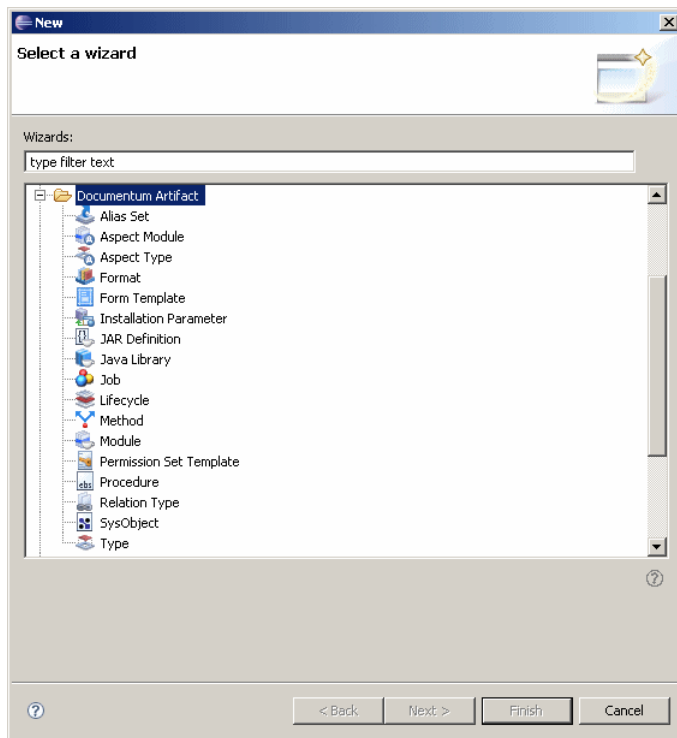
## Creating an artifact

Use the artifact wizard to create an artifact.

### To create an artifact:

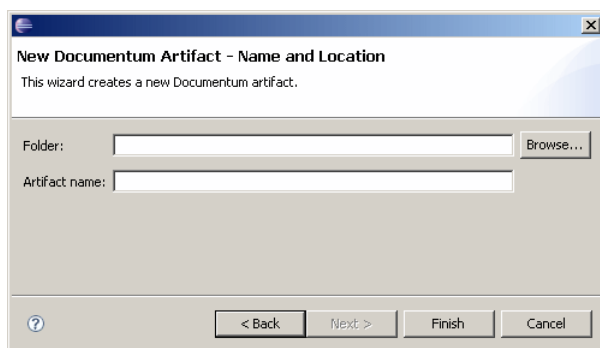
1. Open the **Select a wizard** dialog in one of the following ways:
  - From the Composer menu, select **File > New > Other**.
  - In your project, expand the **Artifacts** folder and right-click the artifact folder in which you want to create an artifact. Select **New > Other**.

The **Select a wizard** dialog appears.



2. Double-click the **Documentum Artifact** folder to expand it, select the artifact you want to create from the artifact list, then click **Next**.

The **New Documentum Artifact – Name and Location** dialog appears.



3. Specify the folder in which you want to create the artifact in the **Folder:** field or accept the default folder.

**Note:** Always create an artifact in the **Artifacts** folder. If you create an artifact directly under the project root, the artifact is not installed properly in the repository.

4. Enter a name for the artifact in the **Artifact name:** field or accept the default artifact name. The default artifact name varies depending on the type of artifact you are creating.

5. Click **Finish**.

The editor for the new artifact appears. For more information about the individual artifact editors and how to configure each artifact's properties, see the associated chapters in this guide.

**Note:** Composer supports copying and pasting of artifacts only within the same project. You cannot copy artifacts from other projects.

## Importing artifacts

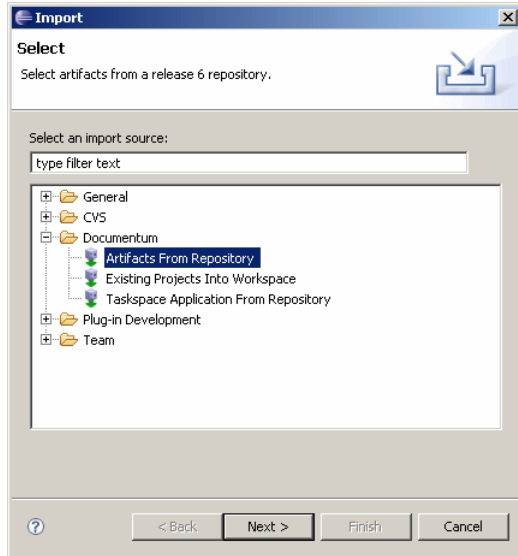
Documentum Composer lets you import individual artifacts from a repository into an existing project. Before importing artifacts, make sure to import and reference all relevant Composer projects that are needed for the artifacts that you are importing. If the artifact that you are importing depends on other artifacts that are not in your project or reference projects, Composer tries to import all other required artifacts from the repository. For more information, see [Composer reference projects, page 25](#).

**Note:** You can only import artifacts from a repository. You cannot import artifacts from a local project into another local project.

### To import individual artifacts:

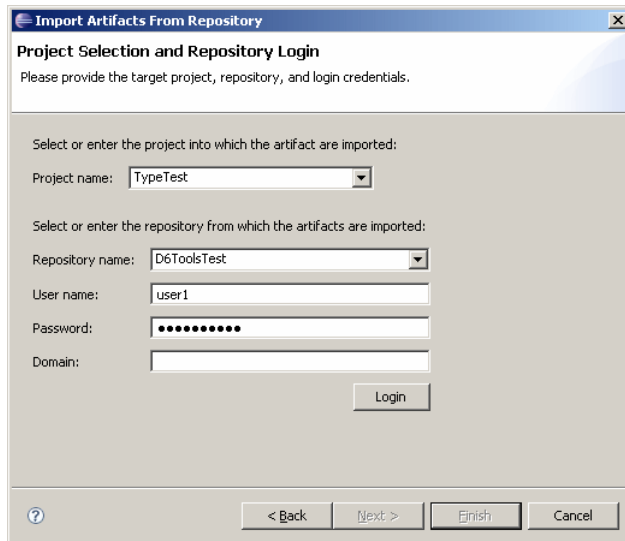
1. Start the **Import** wizard in one of the following ways:
  - Right-click the workspace and select **Import** from the pop-up menu.
  - In the main Eclipse menu, click **File > Import**.

The **Import** wizard appears.



2. Double-click **Documentum** to expand the folder structure, then select **Artifacts From Repository** and click **Next**.

The **Project Selection and Repository Login** dialog appears.



3. Enter the project and repository information, as described in [Table 3, page 32](#), then click **Login**.  
If the login credentials for the repository are correct, you are logged in to the repository.

**Table 3. Project and repository information**

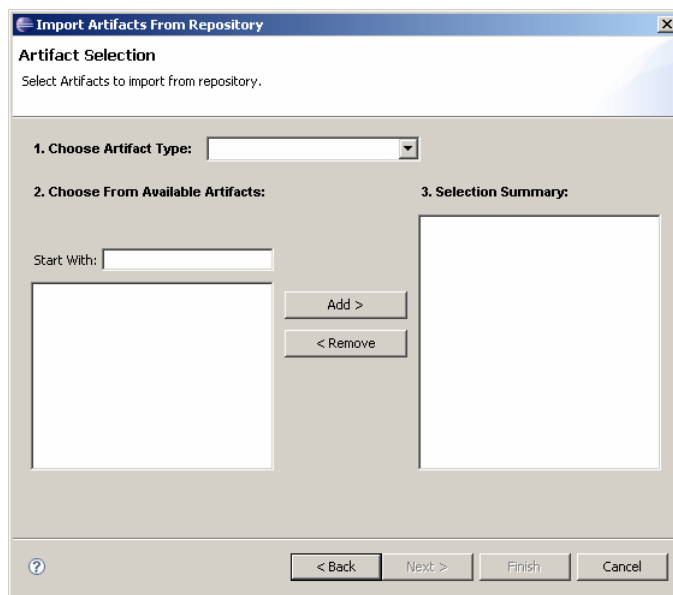
Property	Description
Project name	The name of an existing project into which the artifacts are imported. If you do not have an existing project, create a project before you can import any artifacts. For more information about creating a project, see <a href="#">Creating a project, page 22</a> .
Repository name	The name of the repository that contains the artifacts.



Property	Description
User name	The name used to login to the repository that contains the artifacts.
Password	The password used to login to the repository that contains the artifacts.
Domain	The domain name of the repository. If the repository resides in a different domain than the client from which the repository is accessed, specify the domain name.

- Click **Next >**.

The **Artifact Selection** dialog appears.



- Select the artifact object type from the **Choose Artifact Type** list. The available artifacts of that type appear in the **Available Artifacts** list.

The **Selection Summary** field displays information about the selected artifacts, such as artifact object type and object ID.

- Select one or more objects from the **Available objects** list, then click **Add**.

**Note:** Composer only lists user-defined objects and not objects created by Content Server when a repository is configured.

- When you are done selecting artifacts, click **Finish** to import the artifacts from the repository. The artifacts are imported into the project.

**Note:** If you import an artifact from a repository, move the artifact to a different location within the project, then import the artifact from the repository again, you end up with duplicate artifacts in two different locations in your project.

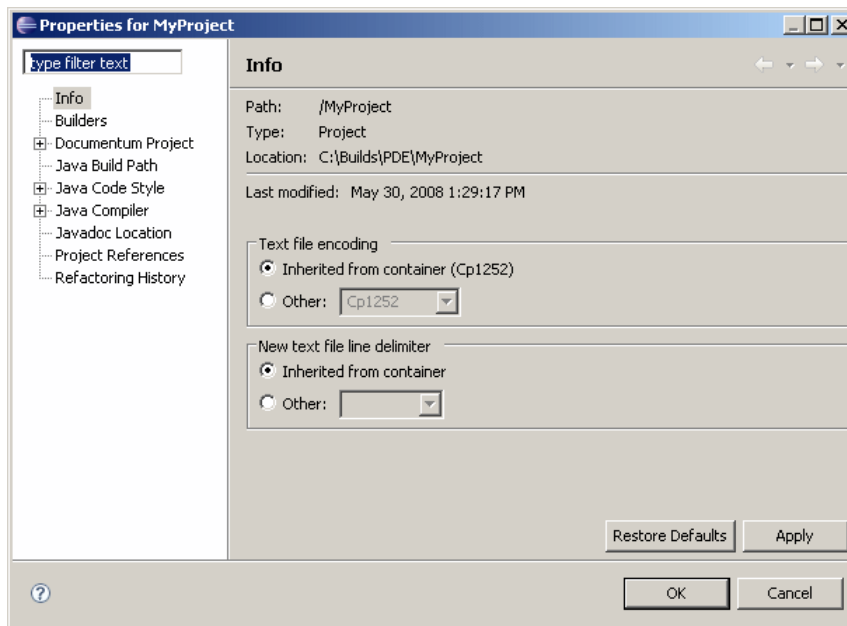
## Configuring project properties

You can configure various project properties using the **Properties** dialog, such as install options, DFS module options, and project install procedures.

To access the **Properties** dialog for a project, right-click the project and select **Properties** from the drop-down list.

The **Properties** dialog appears (Figure 2, page 34).

**Figure 2. Project properties**



For more information about configuring:

- Project install options, see [Configuring the project installation options](#), page 200.
- Project install procedures, see [Configuring pre- and post-installation procedures](#), page 202.
- Project references, see [Designating reference projects for existing Composer projects](#), page 27.
- DFS module options, see [Configuring DFS module options](#), page 65.
- DFS library options, see [Configuring the DFS services library](#), page 66.

## Localizing a Composer project

Composer currently only supports localization of type labels, type attribute labels, UI strings within properties files. Composer also creates a directory structure for BOF JAR files and Java libraries so you can specify localized JAR files if you have them. Composer does not generate the localized JAR files themselves. It only creates the directory structure for you to copy the JAR files into.

Before installing the DAR or Composer project with the localized data dictionary, enable the required locale in the repository. When a repository is created, a set of data dictionary information is loaded,

based on the Content Server host locale. If the host locale is Russian, Arabic, Japanese, Korean, or Simplified Chinese, the data dictionary information for that locale is loaded during repository creation. Otherwise the host locale is always English. To add a new locale, use the population script provided by Documentum. You can populate and publish the data dictionary file by following the procedures located in *Appendix F: Populating and Publishing the Data Dictionary*, in the *Documentum Administrator User Guide*.

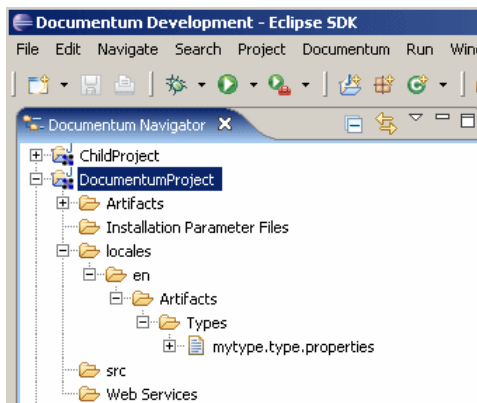
### To localize a Composer project:

1. For every type in the project:
  - a. In the **Attributes** tab, expand an attribute node and click **Application Interface Display**. The **General** section appears to the right.
  - b. In the **General** section, ensure that a value for the **Label** field is specified.
  - c. Complete steps a and b for every attribute.
  - d. In the **Display** tab, ensure that a value for the **Type label** field is specified.

2. In the **Documentum Navigator** view, right-click the project that contains the types that you want to localize.

3. Select **Generate Localization Template** from the drop-down list.

Composer generates a **locales** directory under the project root directory. By default, the **locales** directory contains an English “en” folder, that has the same **Artifacts** directory structure as the main project folder.



The **Artifacts** folder in the **locales** directory lists the artifacts that contain the data that can be localized.

**Note:** You can generate localization templates for locales other than English in Composer 6.6 only.

If you import a project that is a different locale as the version of Composer that you are using, the localizable information in the project is still associated with the locale where it was originally created. The Composer that you are currently using still displays the changes and additions in the original locale. The localization template is also generated in the original locale.

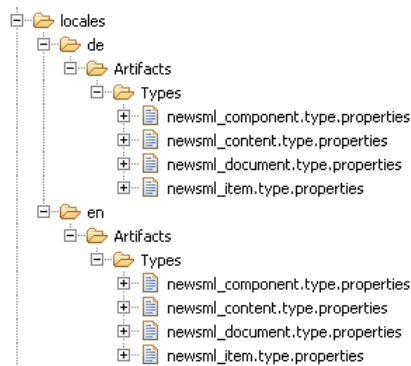
4. If you have any BOF JAR files or Java libraries that require localization, copy the English version of these JAR files into the appropriate directories in the `locales/<lang>/Artifacts` directory. The JAR file must contain the `.properties` files with the localizable strings. For example, if you had a TBO named `my_tbo` in your Composer project and generated a localization template,

a corresponding locales/<lang>/Artifacts/JARs/Modules/TBO/my\_tbo directory would be created so that you can place a localizable JAR file into it. If you do not have any localizable content for your BOF JAR files or Java libraries, delete the directories that you do not need. Composer generates the directories for BOF objects or JAR libraries even if the JAR file does not contain any localizable data. The directories that are created are described in the following list:

- For a Library JAR: locales/<lang>/Artifacts/Java Libraries/<library name>/
- For a standard Module JAR: locales/<lang>/Artifacts/[Standard Module]/<module name>
- For a TBO/SBO Module JAR: locales/<lang>/Artifacts/TBO/<module name>
- For another (typed) Module JAR: locales/<lang>/Artifacts/<module type name>/<module name>

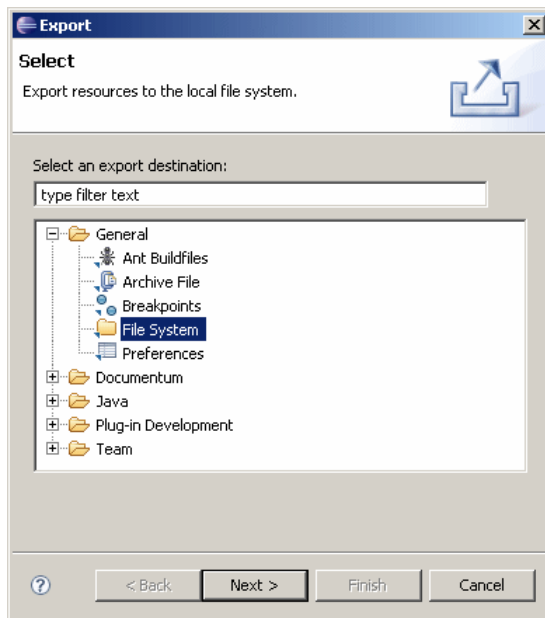
**Note:** When localizing the .properties files in your JAR files, append the locale string to the English .properties filename. For example, if the English .properties filename is localizedStrings.properties, name the localized file localizedStrings\_ja.properties if you are translating to Japanese. Name the localized JAR filename the same as the English JAR filename.

5. Make a copy of the complete **en** folder under the **locales** directory and rename the folder to the language locale you want to make available. For example, if you want to provide German labels for your application, create a **de** folder.

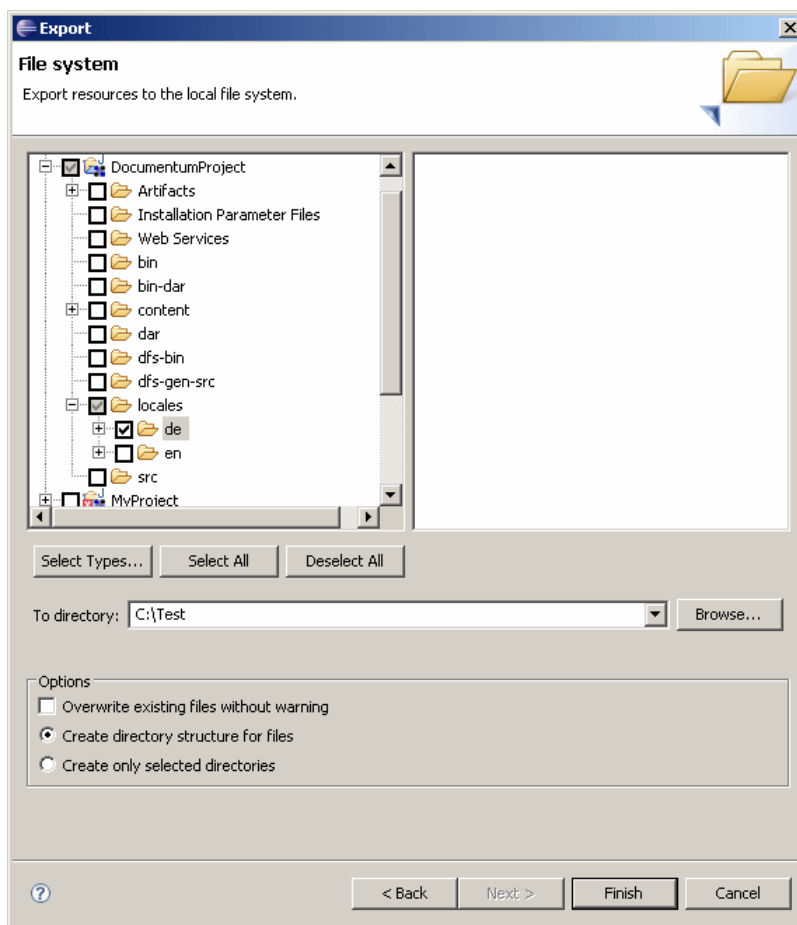


If you are sending out the .properties files for translation, use the following procedure to export the files:

- a. In Composer, select **File > Export**.  
The **Export** dialog appears.



- b. Select **File System**, then click **Next**.  
The **File System** dialog appears.



- c. Expand the project that contains the **locales** folder you want to export, then check the language locales, for example **de**.
- d. Enter the directory to which you want to export the files in the **To directory:** field.
- e. Select the **Create directory structure for files** option, then click **Finish**.

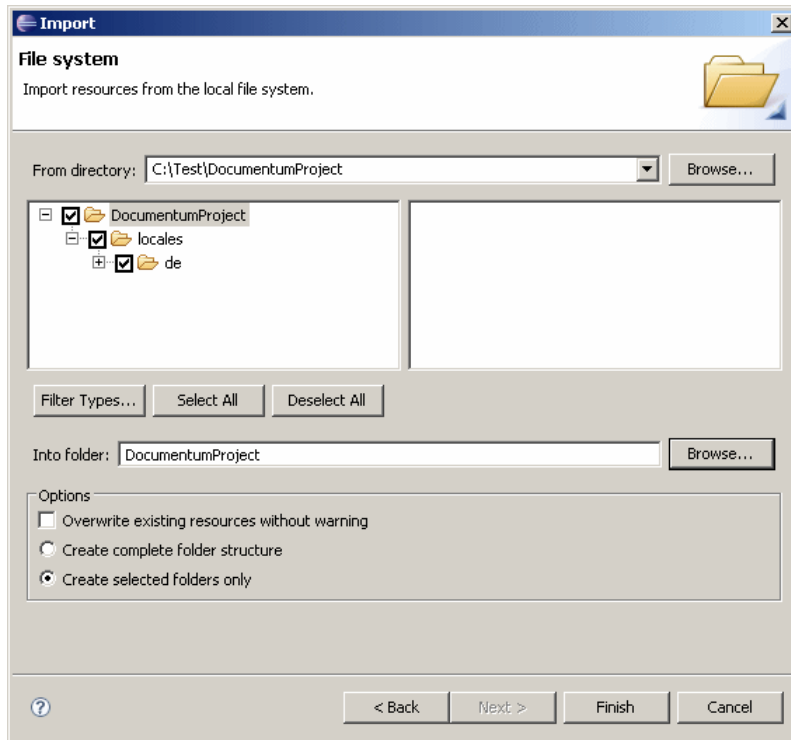
Composer exports the content of the directory structure and content of language locales folder to the selected directory on the local machine. Deliver the files to the translation team.

The translation team translates the strings on the right side of the equal sign (=) in the **.properties** file in the **locales** folder. Do not change the **locales** folder directory structure and **.properties** filenames.

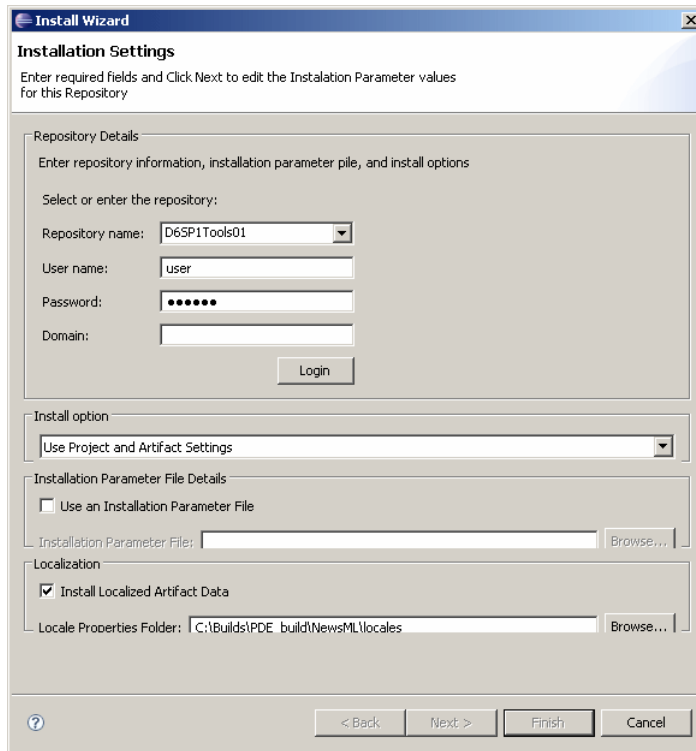


**Note:** The `type/primaryElement/typeAnnotations[0]/locales/value_mappings[xx]/map_data_string` key cannot be localized.

6. After the **.properties** files have been translated, import the files back into the project using the following procedure:
  - a. In Composer, select **File > Import**.  
The Import dialog appears.
  - b. Select **File System**, then click **Next**.  
The **File system** dialog appears.



- c. Enter the directory path to the project folder that contains the **locales** folder with the translated files in the **From directory:** field.
  - d. Select the project and the locales folder.
  - e. Enter the project name of the Documentum project from which the **locales** folder was originally exported in the **Into folder:** field. If the translation team did not change the files names and folder structures, the project name should be identical to the project name in the **From directory** field.
  - f. Select **Create selected folder only**. If you do not want to confirm the import of each file, check **Overwrite existing resources without warning**.
  - g. Click **Finish**.
7. Check the **Install Localized Artifact Data** option in the **Install Wizard** and enter `<Composer_project_root>/locales` as the path to the localized data.



For more information about installing your project, see [Installing a project, page 205](#).

When your project is installed in a repository, the new language strings are automatically merged and the new language becomes available in the repository.

**Note:** We do not recommend changing labels, descriptions, or tabs, or moving the .properties file after you created the localization template because the new language strings might not merge properly. If you rename labels and other localizable application data, regenerate the localization template before you translate the strings.

## Enabling tracing

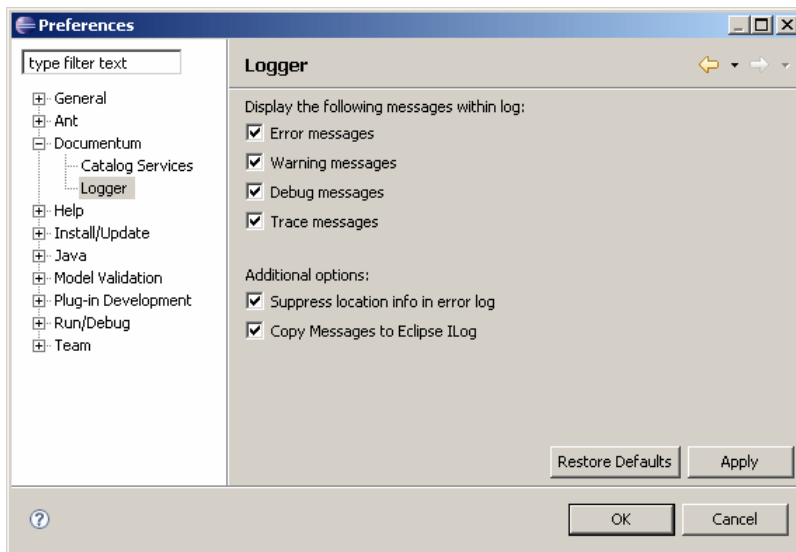
You can enable tracing to monitor processes, for example when building or importing a project. By default, Eclipse has tracing disabled.

**Note:** Use tracing for debugging purposes only, because it impacts Composer performance.

### To enable tracing:

1. In the Eclipse main toolbar, select **Window > Preferences**.  
The **Preference** dialog appears.
2. Double-click Documentum to expand the tree structure and select **Logger**.  
The **Logger** options display.





3. Check the **Trace Messages** and the **Copy Messages to Eclipse ILog** options, then click OK. By default, Composer stores all error log files in the .metadata subdirectory of the workspace.



# Converting DocApps and DocApp archives to Composer projects

DocApps and DocApp archives are Documentum applications that were created using Documentum Application Builder (DAB) prior to release 6.0. With releases 6.0 and later, you create, edit, and install applications with Composer, so convert existing DocApps and DocApp archives to Composer projects if you want to modify them. The conversion is necessary because DocApps and DocApp archives were packaged with proprietary binary files and Composer uses XML files to represent these proprietary binary files.

You can convert DocApps into Composer projects by importing the DocApp straight from the repository. You can also convert DocApp archives into Composer projects by having Composer install the DocApp archive into a repository and converting the project for you.

The following rules apply when converting DocApps and DocApp archives:

- You can convert any existing version 5.3 or later repository DocApp or DocApp archive into a Composer project.
- You can install the resulting Composer project to any 5.3 or later repository. You can install DocApps that you have converted to Composer projects to an older repository as long as the functionality is supported in the older repository. For instance, if your DocApp only has custom subtypes of `dm_document`, you can convert a 6.0 DocApp into a Composer project and install it to a 5.3 repository. However, you cannot do this if the DocApp contained artifacts not supported by Documentum 5.3, such as Smart Containers or Aspects.
- You cannot convert a DocApp or DocApp archive that is stored in a repository prior to version 5.3. If you want to convert, upgrade the DocApp to version 5.3.
- If you are upgrading the repository, do so before converting the DocApp.

## Converting a DocApp to a Composer project

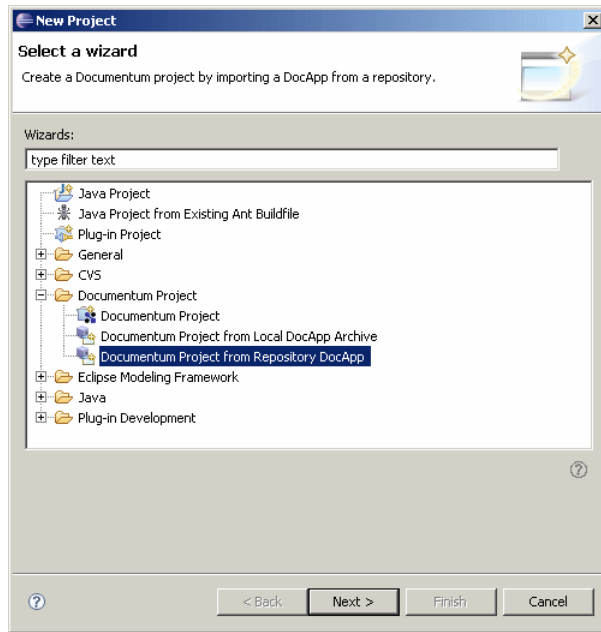
Composer lets you convert a DocApp that is stored in a repository directly into your Composer workspace. This allows you to modify and install your existing DocApps in Composer.

### To convert a DocApp to a Composer project:

1. If the DocApp you are about to convert has dependencies on other DocApps or projects, convert those DocApps first. For example, if the DocApp you are converting uses JAR files that are part of another DocApp, convert the DocApp containing the JAR files first.

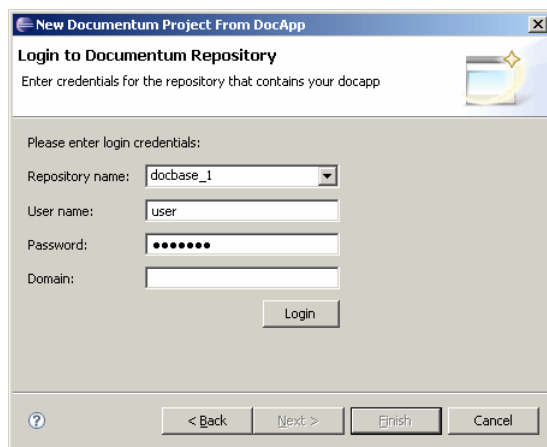
2. Import all Composer projects into the workspace that you want to designate as reference projects. All Documentum supplied reference projects that are needed for your DocApp must be in your workspace before you convert the DocApp. For more information about reference projects, see [Composer reference projects, page 25](#).
3. Open the **New Project** wizard in one of the following ways:
  - From the main menu in Composer select **File > New > Project**.
  - Right-click in the **Documentum Navigator** view and select **New > Project**.

The **New Project** dialog appears.



4. Double-click the **Documentum** folder to expand it, then select **Documentum Project from Repository DocApp** and click **Next**.

The **New Documentum Project from DocApp** dialog appears.



5. Enter your login credentials for the repository that contains the DocApp you want to import, as described in [Table 4, page 45](#), then click **Login**.

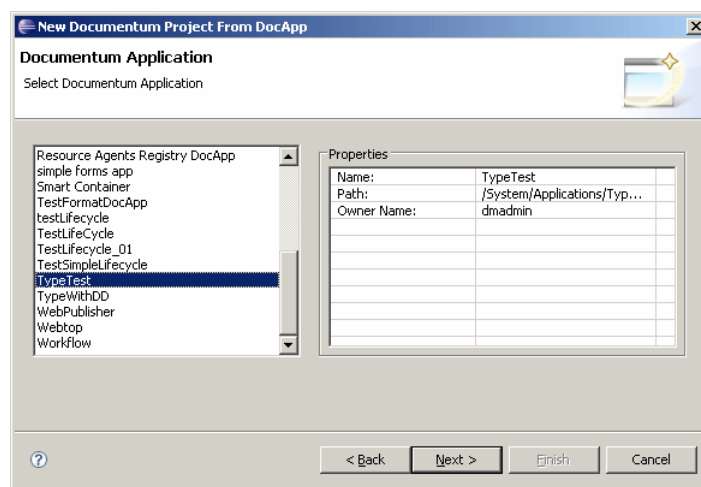
**Table 4. Repository information**

Property	Description
Repository	The name of the repository. Mandatory parameter. You must have SUPERUSER privileges to access the repository.
User name	The user name for the repository.
Password	The password for the repository.
Domain	The domain name of the repository. If the repository resides in a different domain than the client from which the repository is accessed, specify the domain.

Composer connects to the repository and verifies your login credentials.

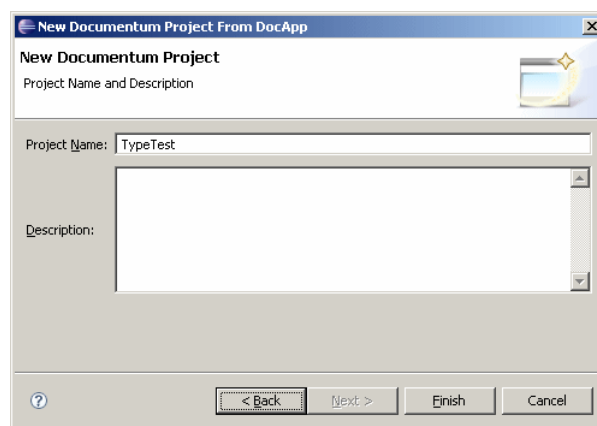
- After your login credentials have been verified, click **Next**.

The **Documentum Application** dialog appears.



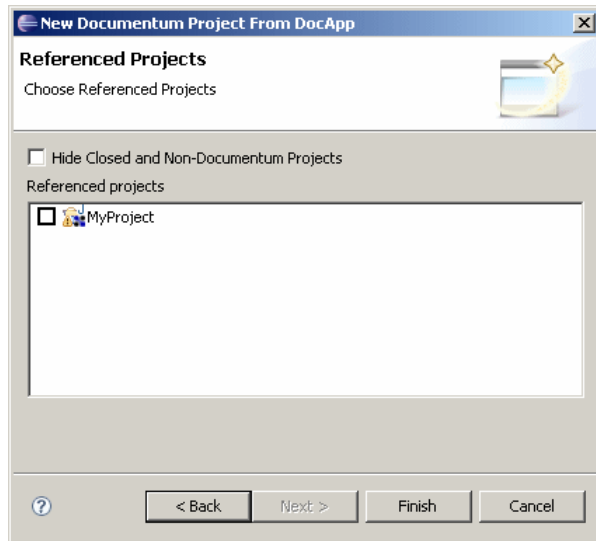
- Select the DocApp you want to convert from the listbox, then click **Next**.

The **New Documentum Project** appears.



- Accept the default project name or enter a new name and an optional description for your project, then click **Next**.

The **Referenced Projects** dialog appears.



9. Select the projects that your project references and click **Finish**.

Composer imports the DocApp and creates a project. The new project is displayed in the **Documentum Navigator** view.

## Converting a DocApp archive to a Composer project

A DocApp archive is a mobile version of a DocApp that was archived using Documentum Application Builder (DAB). To convert a DocApp archive to a Composer project, Composer installs it to a target repository and creates a project from the DocApp archive.

## Preparing for DocApp archive conversion

Before you start converting your DocApp archive into Composer, ensure that you meet the following requirements:

- The target repository for the DocApp archive is a clean repository, meaning it does not contain any remnant DocApps or artifacts. This repository is the repository where you plan on deploying future changes to the resulting Composer project.
- Verify that the DocApp archive you are converting is at least version 5.3. You cannot convert DocApp archives prior to version 5.3. If your DocApp archive is an earlier version than 5.3, upgrade the DocApp archive to version 5.3.
- Verify that the connection broker that is configured in your `dfc.properties` file points to the target repository. For more information about configuring the connection broker, see [Configuring the connection broker, page 18](#).

- Verify that you have SUPERUSER access for the target repository.
- If the DocApp archive you want to convert depends on other DocApps, convert those DocApps and ensure that the resulting Composer project is in your workspace. If your DocApp archive depends on other reference projects, import the reference projects into your workspace as well. See [Composer reference projects](#), page 25 for more information.

## Converting a DocApp archive

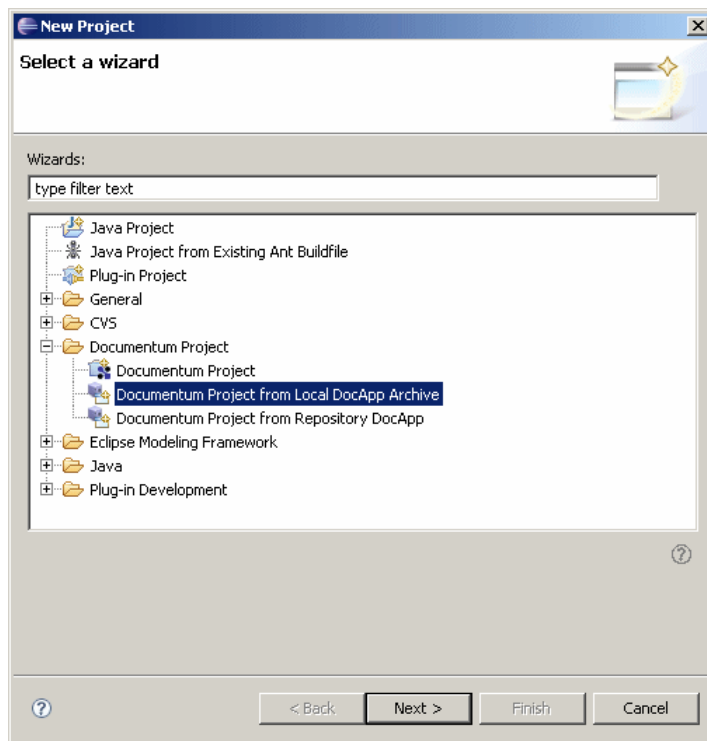
Composer lets you convert a DocApp archive into a Composer project. A DocApp archive is a DocApp that was archived with Documentum Application Builder (DAB).

### To convert a DocApp archive:

1. Unzip the DocApp archive to a folder on your local machine or a network drive.
2. Open the **New Project** wizard in one of the following ways:
  - From the main menu in Composer select **File > New > Project**.
  - Right-click in the **Documentum Navigator** view and select **New > Project**.

The **New Project** dialog appears.

3. Double-click the Documentum folder to expand it, then select **Documentum Project from Local DocApp Archive** and click **Next**.



The **Import Documentum Application Archive** dialog appears.

4. Enter the folder name for the DocApp archive, and a project name, as described in [Table 5, page 48](#), then click **Next**.

**Table 5. DocApp archive import properties**

Parameter	Description
DocApp Archive folder	The folder that contains the unzipped DocApp archive. Mandatory parameter. Type the path name or click <b>Browse</b> to search for the folder.
Project name	The name of the project into which the DocApp is imported. Mandatory parameter. By default, the project name is derived from the DocApp name. You can either accept the default name or enter a new name for the project.

The **Migration Repository Information** dialog appears.

5. Enter the information for the target repository, as described in [Table 6, page 48](#).

**Table 6. Migration repository information**

Parameter	Description
Repository	The name of the target repository. Mandatory parameter. The target repository must be a clean repository. It is used to install the DocApp archive before it is imported into Composer as a new project. You must have SUPERUSER privileges to access the target repository.
User name	The user name for the target repository.



Parameter	Description
Password	The password for the target repository.
Domain	The domain name of the target repository. If the target repository resides in a different domain than the client from which the repository is accessed, specify the domain name.

6. After you have entered the target repository name and your login credentials, click **Login**. If your login credentials are valid, the **Next** button becomes available. Click **Next**.
7. Select any projects that you want to designate as reference projects and click **Finish** to start the conversion process.

Composer creates a project from the DocApp and the project is displayed in the **Documentum Navigator** view.

## Post-conversion tasks

After Composer has converted the DocApp or DocApp archive into a Composer project, complete the following steps:

- Review and correct any validation warnings and errors that occurred during the conversion.
- Verify that all artifacts contained in the DocApp were converted and appear in the Composer project.
- Review pre-install and post-install scripts.

You might have to modify certain scripts to avoid artifact duplication or conflicts. For example, Composer creates install parameters for users. These users must either be created by the pre-install script or must exist in the target repository where the project is going to be installed.



# Composer and the xCelerated Content Platform

EMC Documentum xCelerated Content Platform (xCP) is an application composition platform that provides pre-integrated technologies that combine the power of enterprise content management, business process management, intelligent capture, customer communications management, collaboration, and compliance. It offers rapid application development tools along with deployment best practices to deliver solutions substantially faster and at a lower cost. Because xCP involves composition of services, components, and user interface elements with little or no coding, deployment of a solution requires fewer resources and greatly reduces project risks.

xCP includes the following products:

- Composer
- Content Server
- Business Activity Monitor
- Forms Builder
- Process Builder
- Process Engine/Process Integrator
- TaskSpace
- Process Reporting Services
- Process Analyzer (optional add-on, licensed separately)

In the context of xCP, Composer is used mainly as a packaging and deployment tool for TaskSpace applications or xCP artifacts. TaskSpace applications serve as a container for related xCP artifacts, which can then be imported into Composer as projects and packaged as DAR files. You can also import individual xCP artifacts outside the context of a TaskSpace application into Composer projects and package them as DAR files as well.

# Tips and considerations for packaging and installing TaskSpace applications or xCP artifacts

When packaging and deploying TaskSpace applications or xCP artifacts with Composer, be aware of the following items:

- If a TaskSpace application or xCP artifact changes in the source repository, and you want to repackage it into a Composer project, create a Composer project and do a clean import of the artifacts. Do not use a previously created Composer project that contains old versions of the artifacts.
- You can split up a TaskSpace application into multiple Composer projects to increase portability and manageability. If a part of your application is updated frequently, you can bundle those artifacts into one Composer project and have the rest of the application in another Composer project. If the project contains many artifacts, you can split up TaskSpace applications into multiple Composer projects. For example, it is common to have separate Composer project for types, one for workflows, and another for the containing TaskSpace application. If one of these projects are modified, it is a best practice to repackage and rebuild all of your related Composer projects.
- When importing a Process that uses Task Forms with embedded Forms inside them, the embedded Forms are automatically included in the Composer project.
- When installing a Process that uses Task Forms with embedded Forms inside them, the embedded Forms do not inherit the installation upgrade setting of the Process. For example, if the Process is set to **Version**, the Task Form is automatically set to **Version** regardless of the Task Form's individual settings. The installation upgrade setting is not inherited by any embedded forms inside the Task Form.
- Before installing any project or DAR files that have Form Templates in them, install Process Engine or the FORMS.dar file in the target repository.
- If you are importing a TaskSpace application, some content and artifacts are not imported. Manually import the following items:
  - ACLs and ACL Templates (Composer creates installation parameters for the ACLs and ACL Templates, but does not import the artifacts themselves.)
  - A Group's children groups
  - Groups and Roles that are used in a Workflow Template
  - Alias Sets that are referenced by a Process
  - User content that is not explicitly associated with the TaskSpace application, such as sample documents or libraries of documents.
  - Dynamic Work Queues
  - Custom activity templates that are implemented as BOF modules
  - Necessary .class files for custom workflow methods that are not implemented as BOF modules
  - BOF Modules that are referenced by a method when importing the method

- .class files for custom Form Adapters:
  - Copy the.class file to the App Server if the custom form adapter was built this way. If the custom form adapter was built using BOF modules, then the BOF module and related JAR files are imported automatically.
  - For adapters that use JDBC for connectivity, copy the properties file that contains the JDBC string to the application server.
  - If the Form Adapter fetches data from a properties file, manually import the properties file.
- Custom types that are used in Form Adapters for items such as drop-down menus. It is recommended you add these artifacts to the TaskSpace Application explicitly to make sure that they are imported into Composer:
- The data (objects) in the custom types that are used in Form Adapters
- Registered tables
- Form Instances

## Composer projects and DAR files

You can package TaskSpace applications or xCP artifacts into Composer projects or as DAR files, which are read-only binary representations of a Composer project.

Composer projects are useful if you want to develop and deploy your applications within the Composer IDE. DAR files are useful if you want to decouple development and deployment. A developer would typically hand off a DAR file to an administrator for deployment in this scenario.

The following lists describe the main differences between a DAR file and a Composer project.

### DAR Files

- A single file that is a binary representation of a Composer project. A DAR file must always be generated from a Composer project.
- Cannot be edited
- Cannot be converted into a Composer project
- Generated by the Composer UI or headless Composer
- Installed with the DAR Installer or headless Composer

### Composer Projects

- A set of files that contain the source for a Documentum application.
- Can be edited
- Can be built into a DAR file
- Installed with the Composer UI or headless Composer (must be built into a DAR first).

# Packaging TaskSpace applications

Composer provides a separate plug-in to import Documentum TaskSpace applications as Composer projects. Composer can also package the project into a DAR file, so it can be deployed in other repositories.

## Packaging a TaskSpace application with Composer

Packaging a TaskSpace application involves creating a project in Composer and importing the TaskSpace application. The TCMReferenceProject is now packaged with Composer, so you no longer have to designate this project as a reference project manually.

### To package a TaskSpace application with Composer:

1. Click **File > New Project...** The **New Project** window appears.
2. Select **Documentum Project > Documentum Project from TaskSpace Application** and click **Next**.
3. Select the TaskSpace application that you want to import into Composer from the list on the left and click **Next**.
4. Select any necessary reference projects for your TaskSpace application and click **Finish** to import the TaskSpace application from the repository. After the import, if you are prompted to switch to the Documentum Artifacts perspective, do so.
5. If you want to obtain the DAR file to install with the DAR Installer or headless Composer, complete one of the following steps:
  - If you have the **Project > Build Automatically** option turned on, you can obtain the **<project>.dar** file from the **...\<workspace>\<project-name>\bin-dar** directory.
  - If you have the **Project > Build Automatically** option turned off, right-click the TaskSpace project you want to build and select **Build Project** from the drop-down list. Composer builds the TaskSpace project and generates a **<project>.dar** file in the **...\<workspace>\<project-name>\bin-dar** directory.

## Packaging a TaskSpace application with headless Composer

Headless Composer provides Ant tasks to import TaskSpace applications as Composer projects and to build the Composer project into a DAR file.

### To package a TaskSpace application with headless Composer:

1. Create a build directory to hold the files for your build.
2. In your build directory, create a file named **build.xml**. Open the file for editing.
3. In the **build.xml** file, create a target to create a Composer project with the **emc.createTaskSpaceApplicationProject** task. The task creates a Composer project that has the

same name as the TaskSpace application, and it imports the TaskSpace application from the repository.

4. Create a target to build the project with the emc.build task. Call this task before the emc.dar task to ensure that the DAR file contains the latest built code.
5. Create a target to generate the DAR file with the emc.dar task. The following script shows you how to create the Ant targets for an example project. You can modify the property values at the top of the script for your environment.

```
<?xml version="1.0"?>
<project name="xCPBuild">
  <property name="project.name" value="project_name" />
  <property name="repository" value="repository" />
  <property name="username" value="username" />
  <property name="password" value="password" />
  <property name="dar.filename" value="myDAR.dar" />

  <target name="create-taskpace-project" description="Create a Composer
  project from a TaskSpace application in the repository">
    <emc.createTaskSpaceApplicationProject name="${project.name}" docbase="${repository}
    username="${username}" password="${password}">
    </emc.createTaskSpaceApplicationProject>
  </target>

  <target name="build-project" description="Build the project">
    <emc.build dmproject="${project.name}" failonerror="true"/>
  </target>

  <target name="package-project"
  description="Package the project into a DAR for installation">
    <delete file="${dar.filename}" />
    <emc.dar
      dmproject="${project.name}"
      manifest="bin/dar/default.dardef.artifact"
      dar="${dar.filename}" />
  </target>
</project>
```

6. Create a batch file, build.bat, to run the build. The batch file sets up the Composer workspace and calls the Ant script. When calling the Ant script, call the targets in the exact order as shown in the example. It is important to preserve the order of how the targets are ran. In general, you create the Composer project, import the artifacts into the project, build the project, and then generate the DAR file. The following batch file shows how to run the example build.xml Ant script:

```
REM Set environment variables to apply to this command prompt only
SETLOCAL

REM Sets the root location of headless Composer
SET ECLIPSE="C:\ComposerHeadless"

REM Sets the workspace directory where Composer builds the projects
REM that you want to install to a repository
SET BUILDWORKSPACE=".\\build_workspace"

REM Sets the Ant script that builds your projects
SET BUILDFILE=".\\build.xml"

REM Delete old build and installation workspaces
RMDIR /S /Q %BUILDWORKSPACE%

REM Run Ant scripts to build the project.
REM The JAVA command must be on one line.
```

```
JAVA -cp %ECLIPSE%\startup.jar org.eclipse.core.launcher.Main -data %BUILDWORKSPACE%  
-application org.eclipse.ant.core.antRunner -buildfile %BUILDFILE% create-task-space-project  
build-project package-project
```

7. Run the build.bat file from the command line. When the job completes, the DAR file is output to the location that you specified.

## Packaging xCP artifacts

Packaging xCP artifacts outside the context of a TaskSpace application is useful if you want to separate a TaskSpace application into multiple projects. It is also useful if you want to modify a few xCP artifacts in a TaskSpace application. You can package individual xCP artifacts with Composer or headless Composer.

## Packaging xCP artifacts with Composer

Using the Composer UI allows you to select relevant artifacts for packaging and is useful for non-repeatable or one time packaging of xCP artifacts. This process is no different from importing artifacts from a repository.

### To package xCP artifacts with Composer:

1. Import the desired artifacts as described in [Importing artifacts](#), page 31.
2. If you want to obtain the DAR file to install with the DAR Installer or headless Composer, complete one of the following steps:
  - If you have the **Project > Build Automatically** option turned on, you can obtain the **<project>.dar** file from the **...\<workspace>\<project-name>\bin-dar** directory.
  - If you have the **Project > Build Automatically** option turned off, right-click the TaskSpace project you want to build and select **Build Project** from the drop-down list. Composer builds the TaskSpace project and generates a **<project>.dar** file in the **...\<workspace>\<project-name>\bin-dar** directory.

## Packaging xCP artifacts with headless Composer

Headless Composer provides Ant tasks to create a project and import artifacts directly from a repository. Using the Ant tasks is a helpful way of automating migration of xCP applications from a development environment to a production or QA environment. The following procedure describes how to package an existing xCP application that resides in a repository into a Composer project. It also describes how to generate a DAR file that can be deployed on another system.

### To package xCP artifacts with headless Composer:

1. Create a build directory to hold the files for your build.
2. In your build directory, create a file named build.xml. Open the file for editing.



3. In the build.xml file, create a target to create a Composer project with the emc.createArtifactsProject task.
4. Create a target to import the desired xCP artifacts into the project with the emc.importArtifacts task.
5. Create a target to build the project with the emc.build task. Call this task before the emc.dar task to ensure that the DAR file contains the latest built code.
6. Create a target to generate the DAR file with the emc.dar task. The following script shows you how to create the Ant targets for an example project. You can modify the property values at the top of the script for your environment.

```
<?xml version="1.0"?>
<project name="xCPBuild">
  <property name="project.name" value="project_name" />
  <property name="repository" value="repository" />
  <property name="username" value="username" />
  <property name="password" value="password" />
  <property name="artifact.path"
    value="/System/Applications/app_name/artifact_name" />
  <property name="dar.filename" value="myDAR.dar" />

  <target name="create-project" description="Create a project to import
    artifacts into">
    <emc.createArtifactProject name="${project.name}" overwrite="true">
    </emc.createArtifactProject>
  </target>

  <target name="import-project" description="
    Must import a project before updating, building, or installing it">
    <emc.importProject dmpproject="HelloWorldArtifacts" failonerror="true"/>
  </target>

  <target name="import-artifacts">
    <emc.importArtifacts project="${project.name}" docbase="${repository}"
      username="${username}" password="${password}">
      <objectIdentities>
        <path value="${artifact.path}"/>
      </objectIdentities>
    </emc.importArtifacts>
  </target>

  <target name="build-project" description="Build the project">
    <emc.build dmpproject="${project.name}" failonerror="true"/>
  </target>

  <target name="package-project"
    description="Package the project into a DAR for installation">
    <delete file="${dar.filename}" />
    <emc.dar
      dmpproject="${project.name}"
      manifest="bin/dar/default.dardef.artifact"
      dar="${dar.filename}" />
  </target>

</project>
```

7. Create a batch file, build.bat, to run the build. The batch file sets up the Composer workspace and calls the Ant script. When calling the Ant script, call the targets in the exact order as shown in the example. It is important to preserve the order of how the targets are ran. In general, you create the Composer project, import the artifacts into the project, build the project, and then generate the DAR file. The following batch file shows how to run the example build.xml Ant script:

```
REM Set environment variables to apply to this command prompt only
SETLOCAL

REM Sets the root location of headless Composer
SET ECLIPSE="C:\ComposerHeadless"

REM Sets the workspace directory where Composer builds the projects
REM that you want to install to a repository
SET BUILDWORKSPACE=".\\build_workspace"

REM Sets the Ant script that builds your projects
SET BUILDFILE=".\\build.xml"

REM Delete old build and installation workspaces
RMDIR /S /Q %BUILDWORKSPACE%

REM Run Ant scripts to build the project.
REM The JAVA command must be on one line.
JAVA -cp %ECLIPSE%\startup.jar org.eclipse.core.launcher.Main -data %BUILDWORKSPACE%
-application org.eclipse.ant.core.antRunner -buildfile %BUILDFILE% create-project
import-artifacts build-project package-project
```

When the build.bat file is done running, a DAR file appears in the location that you specified for the `{dar.filename}` property. You can deploy the DAR with the DAR Installer or with headless Composer.

## Installing TaskSpace applications and xCP artifacts

Installing TaskSpace applications and xCP artifacts follow the same process as any other Composer project. You can use either the Composer UI to install Composer projects, the DAR Installer to install DAR files, or headless Composer to install both Composer projects and DAR files.

### Installing a TaskSpace application and xCP artifacts with Composer

When you have a TaskSpace application successfully imported into Composer as a project, you can use Composer to install the project into a repository. The process is the same as installing any other Composer project as described in [Installing a project, page 205](#).

### Installing TaskSpace applications and xCP artifacts with the DAR Installer

If you have a TaskSpace application DAR file or xCP DAR file, you can install it with the DAR Installer. The process is the same as installing any other Composer DAR file as described in [Installing a DAR file with the DAR Installer, page 209](#). You no longer need to copy the TCMReferenceProject.dar file to the same directory as your DAR file for installation.

## Installing TaskSpace applications and xCP artifacts with headless Composer

To deploy TaskSpace applications or xCP artifacts with headless Composer, package the application or artifacts into a DAR file as described in the following sections:

- [Packaging a TaskSpace application with Composer, page 54](#)
- [Packaging a TaskSpace application with headless Composer, page 54](#)
- [Packaging xCP artifacts with Composer, page 56](#)
- [Packaging xCP artifacts with headless Composer, page 56](#)

When you have built a DAR file from a Composer project, you can install the DAR file with the `emc.install` task.

### To deploy xCP artifacts with headless Composer:

1. Create a build directory to hold the files for your build.
2. Create a file named `install.xml`.
3. Create a target to install a DAR file with the `emc.install` task. The following script is an example of how to declare the `emc.install` task.

#### Example 4-1. Example `install.xml` script

```
<?xml version="1.0"?>
<project name="headless-install">

  <property name="repository" value="repository" />
  <property name="username" value="username" />
  <property name="password" value="password" />
  <property name="dar.filename" value="myDAR.dar" />

  <target name="install-project"
    description="Install the project to the specified repository.
    dfc.properties must be configured">
    <emc.install
      dar="${dar.filename}"
      docbase="${repository}"
      username="${username}"
      password="${password}"
      domain="" />
    </target>

  </project>
```

4. Create a batch file, `install.bat`, to run the installation script. The batch file sets up the Composer workspace and calls the Ant script. When calling the Ant script, call the targets in the exact order as shown in the example. The following batch file shows how to run the example `install.xml` Ant script:

```
REM Set environment variables to apply to this command prompt only
SETLOCAL

REM Sets the root location of headless Composer
SET ECLIPSE="C:\ComposerHeadless"

REM Sets the workspace directory where Composer extracts built DAR files
```

```
REM before installing them to a repository
SET INSTALLWORKSPACE=".\\install_workspace"

REM Sets the Ant script that builds your projects
SET INSTALLFILE=".\\install.xml"

REM Delete old build and installation workspaces
RMDIR /S /Q %INSTALLWORKSPACE%

REM Run Ant scripts to install the project
REM The JAVA command must be on one line.
JAVA -cp %ECLIPSE%\startup.jar org.eclipse.core.launcher.Main -data %INSTALLWORKSPACE%
-application org.eclipse.ant.core.antRunner -buildfile %INSTALLFILE% install-project
```

## Building and installing Composer projects that already contain xCP artifacts with headless Composer

If you have a Composer project that already contains xCP artifacts, and you want to build and deploy the project, headless Composer allows you to automate this procedure.

### To build and deploy a Composer project that contains xCP artifacts:

1. Create a build directory to hold the files for your build.
2. In your build directory, create a file named build.xml. Open the file for editing.
3. In the build.xml file, create a target to import the Composer project that contains the xCP artifacts with the emc.import task.
4. Create a target to build the project with the emc.build task. Call this task before the emc.dar task to ensure that the DAR file contains the latest built code.
5. Create a target to generate the DAR file with the emc.dar task. Call this task before the emc.install task to ensure that the code built from emc.build task makes it into the new DAR file.
6. Create a target to install the DAR file with the emc.install task. The following script shows you how to create the Ant targets for an example project. You can modify the property values at the top of the script for your environment.

```
<?xml version="1.0"?>
<project name="xCPBuild">
  <property name="project.name" value="project_name" />
  <property name="repository" value="repository" />
  <property name="username" value="username" />
  <property name="password" value="password" />
  <property name="dar.filename" value="myDAR.dar" />

  <target name="import-project" description="
  Must import a project before updating, building, or installing it">
    <emc.importProject dmpproject="${project.name}" failonerror="true"/>
  </target>

  <target name="build-project" description="Build the project">
    <emc.build dmpproject="${project.name}" failonerror="true"/>
  </target>
```

```

<target name="package-project"
  description="Package the project into a DAR for installation">
  <delete file="${dar.filename}" />
  <emc.dar
    dmproject="${project.name}"
    manifest="bin/dar/default.dardef.artifact"
    dar="${dar.filename}" />
</target>

<target name="install-project"
  description="Install the project to the specified repository.
  dfc.properties must be configured">
  <emc.install
    dar="${dar.filename}"
    docbase="${repository}"
    username="${username}"
    password="${password}"
    domain="" />
</target>

</project>

```

7. Create a batch file, build.bat, to run the build. The batch file sets up the Composer workspace and calls the Ant script. When calling the Ant script, call the targets in the exact order as shown in the example. It is important to preserve the order of how the targets are ran. In general, you create the Composer project, import the artifacts into the project, build the project, and then generate the DAR file. The following batch file shows how to run the example build.xml Ant script:

```

REM Set environment variables to apply to this command prompt only
SETLOCAL

REM Sets the root location of headless Composer
SET ECLIPSE="C:\ComposerHeadless"

REM Sets the location of your source projects.
REM This location gets copied into your build workspace directory
SET PROJECTSDIR="C:\Documents and Settings\Administrator\composer-workspace"

REM Sets the workspace directory where Composer builds the projects
REM that you want to install to a repository
SET BUILDWORKSPACE=".\\build_workspace"

REM Sets the workspace directory where Composer extracts built DAR files
REM before installing them to a repository
SET INSTALLWORKSPACE=".\\install_workspace"

REM Sets the Ant script that builds your projects
SET BUILDFILE=".\\build.xml"

REM Delete old build and installation workspaces
RMDIR /S /Q %BUILDWORKSPACE%
RMDIR /S /Q %INSTALLWORKSPACE%

REM Copy source projects into build workspace
XCOPY %PROJECTSDIR% %BUILDWORKSPACE% /E

REM Run Ant scripts to build and install the projects
REM Each JAVA command must be on one line.
JAVA -cp %ECLIPSE%\startup.jar org.eclipse.core.launcher.Main -data %BUILDWORKSPACE%
-application org.eclipse.ant.core.antRunner -buildfile %BUILDFILE%
import-project build-project package-project
JAVA -cp %ECLIPSE%\startup.jar org.eclipse.core.launcher.Main -data %INSTALLWORKSPACE%
-application org.eclipse.ant.core.antRunner -buildfile %BUILDFILE% install-project

```

# Migrating a TaskSpace application or xCP artifacts from a source environment to a target environment

Documentum Composer allows you to migrate TaskSpace applications or individual xCP artifacts from a source environment to a target environment. This procedure is only valid if the source and target environment contain their own xCP stack (including Content Server, Process Engine, TaskSpace, and Composer). An example of this scenario is migrating a project from a development environment to a production environment.

**Note:** To migrate a TaskSpace application or xCP artifacts from a D6.5 SP2 environment to a D6.6 environment, you must follow this procedure.

## Packaging the TaskSpace application or xCP artifacts on the source environment

Packaging your application involves building the artifacts into a Composer DAR file.

Before beginning this procedure, the following prerequisites must be met:

- The target environment must be D6.6
- Composer version 6.6 must be installed even if the source environment version is older than version 6.6. As a best practice, use the same version of Composer for both packaging and deployment.
- The `dfc.properties` file must point to a valid connection broker. The file is located in the folder `<Composer_root>\plugins\com.emc.ide.external.dfc_1.0.0\documentum.config`. For more information on configuring the connection broker, see [Configuring the connection broker, page 18](#)

### To migrate your TaskSpace application, package it as a DAR:

1. Start Composer D6.6 and create a workspace.
2. Create a Composer project from a TaskSpace application as described in [Packaging a TaskSpace application with Composer, page 54](#). If you want to migrate individual xCP artifacts, import them as described in [Importing artifacts, page 31](#).
3. If you have sample content that you want to migrate, such as folders or individual documents, import them individually as described in [Importing artifacts, page 31](#)  
Composer automatically builds the project into a DAR file that you can use to install to the target repository. You can find the DAR file in the `<workspace_root>/<project_name>/bin-dar` directory. If you do not see a DAR file, read the [Generating a DAR file, page 205](#) section to see how to generate one.
4. If the source environment cannot communicate with the target environment, zip the entire Composer project directory, `<workspace_root>/<project_name>`, and transfer it to the target environment.

**Note:** If you want to improve performance, transferring the Composer project directory to the target environment also decreases installation time.

## Deploying the TaskSpace application or xCP artifacts on the target repository

When you have the DAR file ready, you can deploy it to the target environment. Before beginning this procedure, Composer version 6.6 must be installed.

### To deploy the TaskSpace application or xCP artifacts:

1. If you transferred the Composer project zip file to the target environment:
  - a. Unzip the Composer project that contains the TaskSpace application or xCP artifacts to a directory of your choice.
  - b. Copy the <Composer\_project>/bin-dar/<xCP\_app>.dar to a directory of your choice.
2. Run <Composer\_root>\darinstaller.exe.
3. Install the DAR file with the DAR Installer.
4. If you migrated a TaskSpace application:
  - a. Access the TaskSpace application by going to [http://host:port/Taskspace/?appname=<TaskSpace\\_app\\_name>](http://host:port/Taskspace/?appname=<TaskSpace_app_name>) using a TaskSpace administrator username and password.
  - b. Navigate to the Administration tab.
  - c. Create users if desired.
  - d. Assign users to roles within the TaskSpace application.

## Troubleshooting tips

If you receive an error regarding Presets after trying to access the TaskSpace application on the target environment(), it was not installed correctly.

### To fix the Presets error:

1. Log in to Documentum Administrator.
2. Go to the folder System /Applications/<TaskSpace\_application>
3. Verify that the owner of the following artifacts is "dmc\_wdk\_presets\_owner":
  - Presets
  - Presets / TaskSpace\_App.definition
  - Presets / TaskSpace\_Role.definition
  - Presets / Preset Packages
  - Files within the folder Presets / Preset Packages

If not, then set the owner of the artifacts to "dmc\_wdk\_presets\_owner".



# Managing Web Services

This chapter contains the following topics:

- [Web services, page 65](#)
- [Configuring DFS module options, page 65](#)
- [Configuring the DFS services library, page 66](#)
- [Configuring catalog services, page 67](#)
- [Viewing Web services, page 69](#)
- [Generating a client proxy , page 71](#)
- [Creating a service, page 73](#)
- [Modifying catalog and category information, page 75](#)
- [Publishing a service , page 76](#)
- [Unpublishing a service, page 77](#)
- [Exporting a service, page 78](#)
- [Deploying a service, page 79](#)

## Web services

On a high level, a Web service is a software system designed to support interoperable machine-to-machine interaction over a network. Web services are often Web APIs that can be accessed over a network, such as the Internet, and executed on a remote system hosting the requested services.

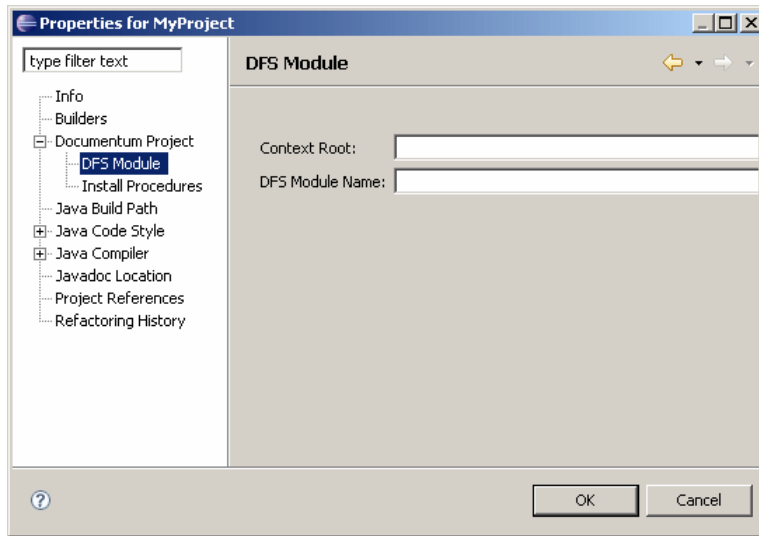
Composer supports Web services by providing an EMC Documentum Foundation Services (DFS) registry plug-in. The plug-in allows users to connect to a Web services registry, import WSDLs to create a Java client library, create services, and export the services to an EAR file. Composer includes a DFS Builder and a DFS Services Library for each new Documentum project. The DFS Builder and DFS Service Library can be configured in a project's property settings.

## Configuring DFS module options

You can configure the DFS context root and module name for a project in the **DFS Module** dialog.

**To configure the DFS module options:**

1. Right-click the project and select **Properties** from the drop-down list.  
The **Properties** dialog appears (Figure 2, page 34).
2. Expand **Documentum Project** and select **DFS Module**.  
The **DFS Module** dialog appears.



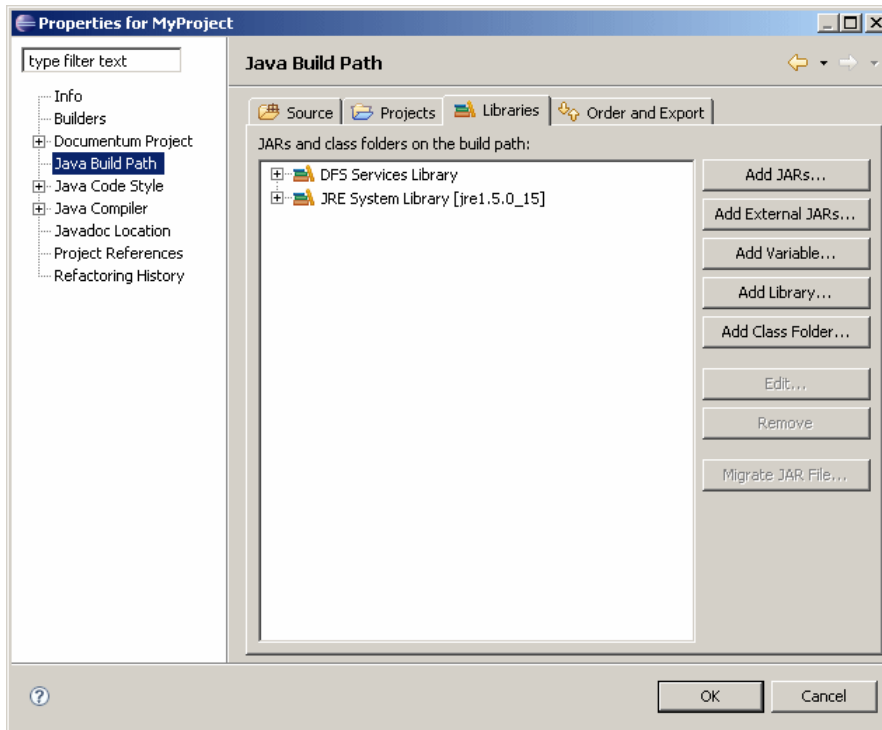
3. Enter the DFS context root and module name in the **Context Root** and **DFS Module Name** fields.
4. Click OK.

## Configuring the DFS services library

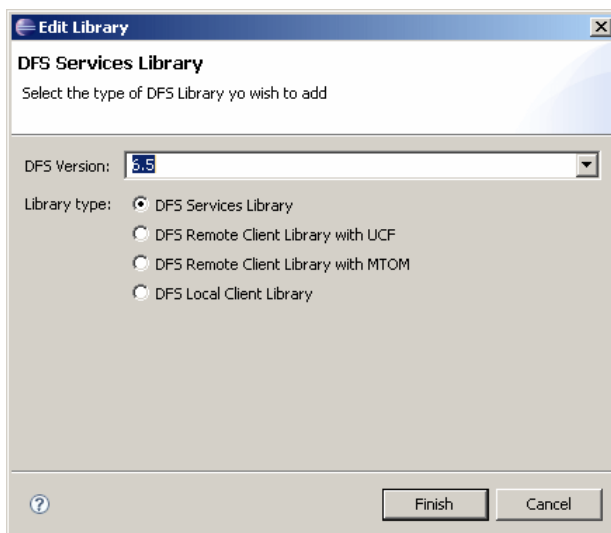
Composer lets you select the DFS service library you want to use for a project. The DFS service library is configured in the project properties. By default, Composer is shipped with one DFS services library, but can support multiple DFS services libraries.

**To configure the DFS services library:**

1. Right-click the project and select **Properties** from the drop-down list.  
The **Properties** dialog appears (Figure 2, page 34).
2. Select **Java Build Path**.  
The **Java Build Path** dialog appears.



3. Click the **Libraries** tab, select **DFS Services Library** from the list box and click **Edit**. The **DFS Services Library** dialog appears.



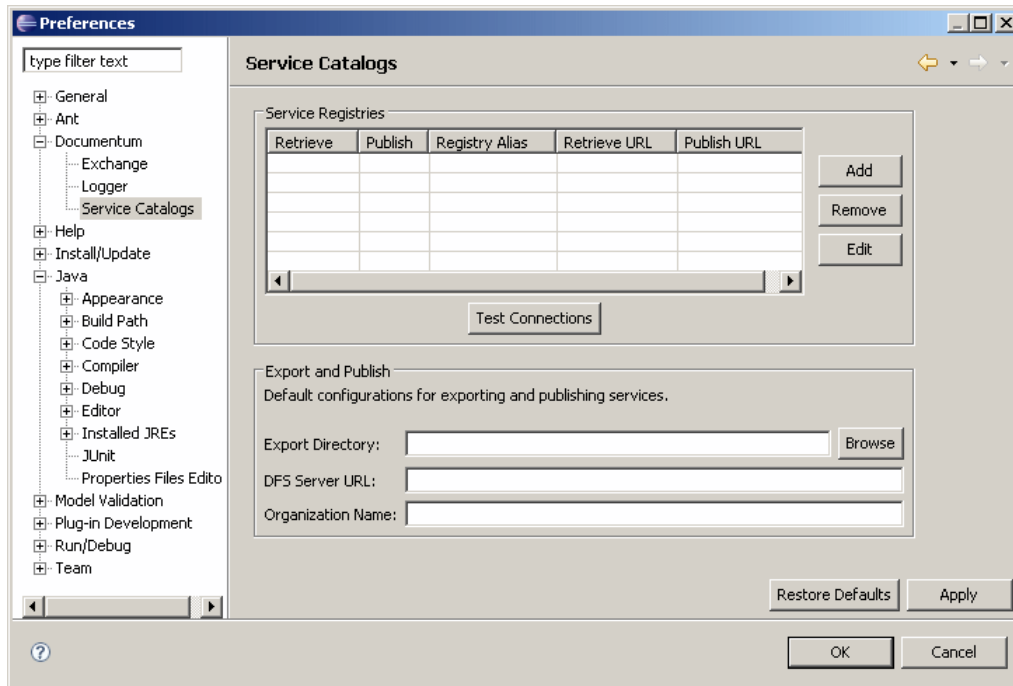
4. Select the type of DFS library you wish to add, then click **Finish**.

## Configuring catalog services

The catalog services connection options are configured in the **Preferences** dialog.

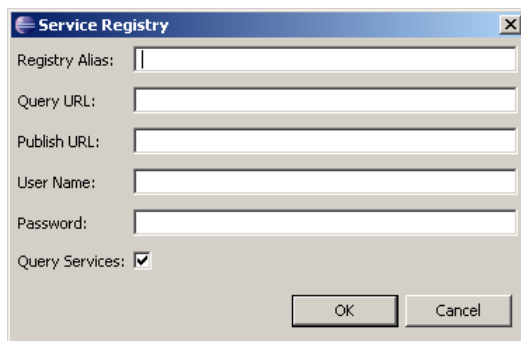
### To configure catalog services:

1. In the Composer main menu, select **Window > Preferences ...**  
The **Preferences** dialog appears.
2. In the **Preferences** list, double-click **Documentum** and select **Catalog Services**.  
The **Catalog Services** dialog appears.



The **Services Catalogs** table lists the services that are currently configured.

3. To configure another service registry, click **Add**.  
The **Service Registry** dialog appears.



4. Enter the configuration parameters for the service registry, as described in [Table 7, page 68](#).

**Table 7. Service registry options**

Parameter	Description
Registry Alias	A string specifying a name for the service registry.

Parameter	Description
Query URL	A string specifying the URL of the server that hosts the service. The URL must have the format <code>http://&lt;domain&gt;:&lt;port&gt;/catalog/inquiry</code> .
Publish URL	A string specifying the URL of the server to which the service is published. The URL must have the format <code>http://&lt;domain&gt;:&lt;port&gt;/catalog/publish</code> .
User Name	The login name for the server hosting the service.
Password	The password for the server hosting the service.

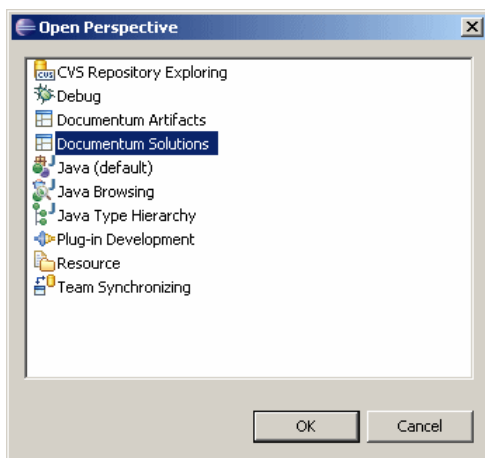
## Viewing Web services

In Composer all Web services and related actions are displayed in a different perspective, the **Documentum Solutions** perspective.

### To view available Web services:

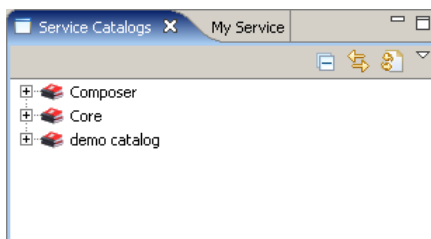
1. In the Composer main menu, select **Window > Open Perspective > Other ...**

The **Open Perspective** dialog appears.



2. Select **Documentum Solutions** and click **OK**.

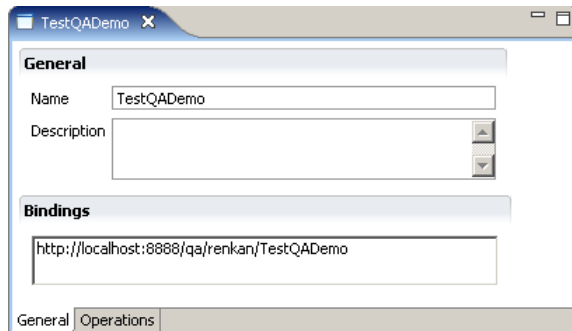
The **Service Catalog** tab and the **My Service** tab display.



The **Service Catalog** tab shows all service catalogs and services on the server that you configured. The **MyService** tab shows the services you imported or created. If you installed the Documentum Services Catalog Repository, Documentum services are not automatically published as part of that installation. Services must be published to the catalog before you can view them in Composer. If no services are published, Composer does not display any services.

3. To view the Web services, expand the catalog, then double-click the service to display the service details.

The service details appear with the **General** tab selected.



4. Click the **Operations** tab to view the service methods.

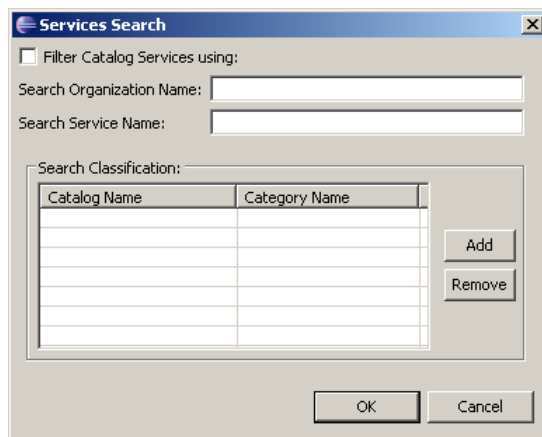
## Filtering services

You can specify which services you want to display in the **Service Catalog** tab by using the service filter.

### To filter services:

1. In the **Documentum Solutions** perspective, click the search icon (🔍) below the **Catalog Services** tab.

The **Services Search** dialog appears.



2. Select **Filter Catalog Services using:** and enter your search criteria. You can filter by **Organization Name**, **Service Name**, **Catalog Name**, and **Category Name**.

3. Click **OK**.

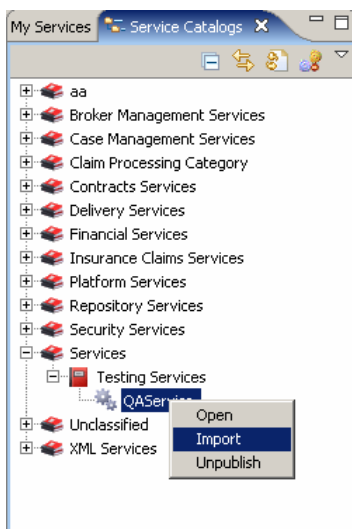
The services that match the filter criteria are displayed in the **Catalog Services** view.

## Generating a client proxy

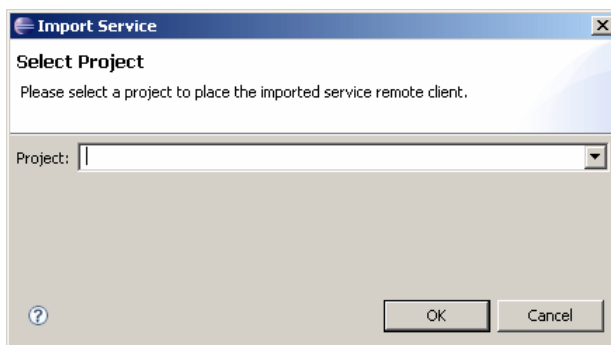
The **Import** option in the **Service Catalog** tab lets you generate the client proxy of a service and make it available in a Composer project.

### To generate the client proxy of a service:

1. In the Service Catalog view, right-click the service you want to generate the client proxy and select **Import**.

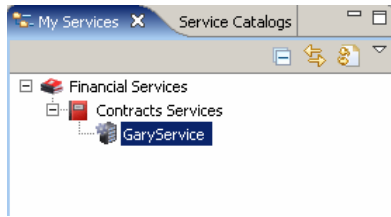


The **Import Service** dialog appears.

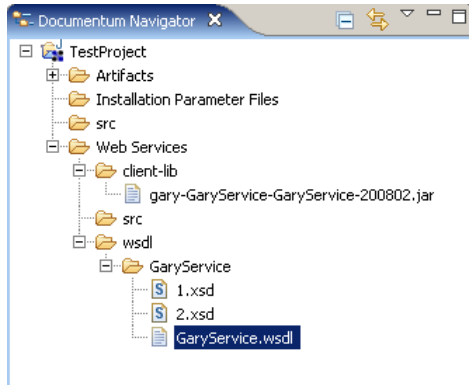


2. Enter the project in which you want to generate the client proxy or select a project from the drop-down list, then click **OK**.

Composer imports the client proxy into the project. The service name appears in the **My Services** tab.



The JAR file and the WSDL of the service appear in the **Web Services** folder of the project in the **Documentum Navigator** view.



## Consuming a service

Consuming a service requires importing the client proxy of a service, as described in [Generating a client proxy](#), page 71, and creating the code that calls the service.

The following code example describes how to call a service. The only custom code in the example is the **try** block that is highlighted in bold.

```
package com.acme.loanapp.services;

import com.emc.documentum.fs.datamodel.core.context.RepositoryIdentity;
import com.emc.documentum.fs.rt.context.ContextFactory;
import com.emc.documentum.fs.rt.context.IServiceContext;
import com.emc.documentum.fs.rt.context.ServiceFactory;
import com.emc.services.ws.client.soap.*;

public class AcmeLoanServiceOrchestration
{
    public static void main(String [ ] args)
    {
        RepositoryIdentity m_theId = new RepositoryIdentity();

        m_theId.setRepositoryName("D65Docbase");
        m_theId.setUserName("dfsuser");
        m_theId.setPassword("dfs");

        //completion point 'get context'
        IServiceContext context = ContextFactory.getInstance().newContext();
        ServiceFactory sf = ServiceFactory.getInstance();
    }
}
```



```

        context.addIdentity(m_theId);

    try {
        //completion point 'instantiate services'
        IWorkflowService qSvc = sf.getRemoteService(IWorkflowService.class,
            context, "core", "http://localhost:9080/services");
        qSvc.start("ProcessLoanApplication");
    }
    catch (Exception e)
    {
        System.out.println("An exception has occurred: " + e);
    }
}

```

## Creating a service

You can create a service from a Java file or generate a service from the WSDL file of a client proxy.

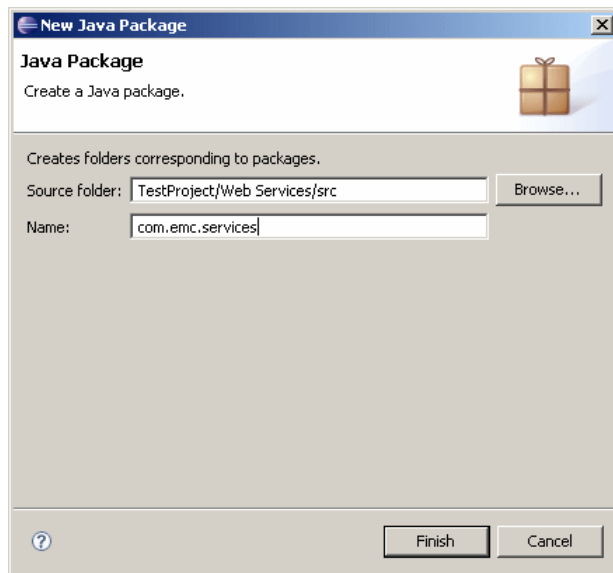
## Creating a service from a Java file

You must switch to the **Package Explorer** view to create a service.

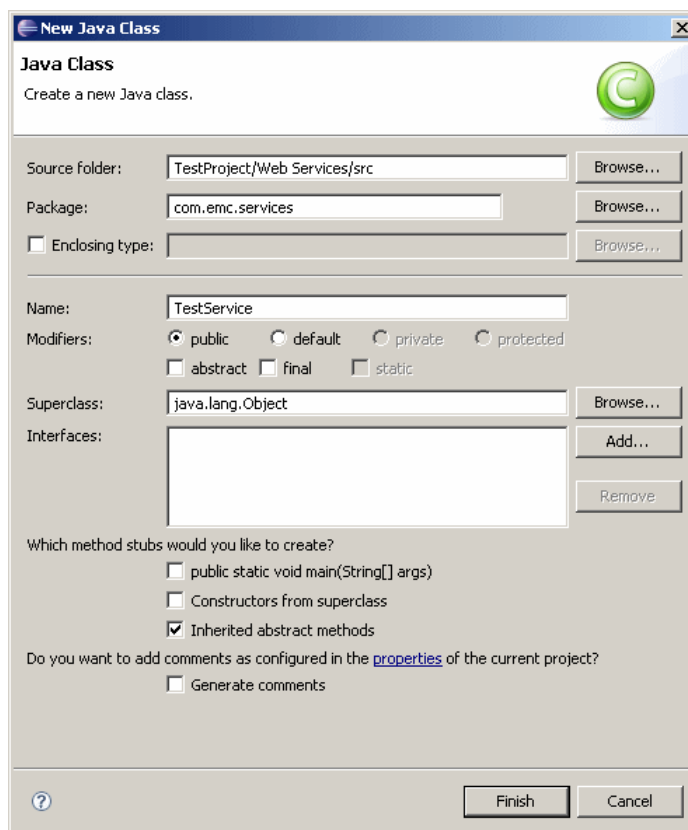
**Note:** This guide only describes how to create the Java file for the service, not how to develop a service. For more information about developing DFS services, see the *EMC Documentum Documentum Foundation Services Development Guide Version 6.5*.

### To create a service from a Java file:

1. Change to the **Package Explorer** view in Composer by selecting **Window > Show View > Package Explorer**.
2. Create a Java package for your service in the **Web Service/src** directory:
  - a. Right-click the **Web Service/src** directory and select **New > Package**.  
The **New Java Package** dialog appears.



- b. Enter a name for your Java package, for example **com.emc.services**, then click **Finish**.
3. Create a Java class:
  - a. Right-click the Java package that you created and select **New > Class**.  
The **New Java Class** dialog appears.



- b. Enter a name for your Java class, for example **TestService** and select any methods you want to include, then click **Finish**. The Java file appears in the workspace.

- c. Write the code that specifies your service. For more information about developing DFS services, see *EMC Documentum Documentum Foundation Services Development Guide Version 6.5*.
- d. Save your changes.  
Your new service appears on the **MyServices** tab in the **Documentum Solutions** perspective under **Unclassified**.

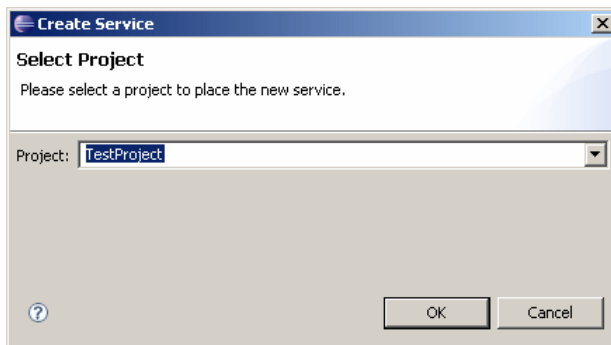
## Creating a service from a WSDL

You can create a service directly from a WSDL file.

### To create a service from a WSDL:

1. Navigate to the Web Services folder of the project in the **Documentum Navigator** view.
2. Right-click the WSDL from which you want to generate a service and select **Create Service** from the drop-down menu.

The **Create Service** dialog appears.



3. Enter the name of the project in which you want Composer to create the service or select a project from the drop-down list, then click **OK**.

The Java file for the service appears in the `/src` directory of the project's **Web Services** folder.

## Modifying catalog and category information

When you first create a service, as described in [Creating a service, page 73](#), the service appears under the **Unclassified** catalog and category in the **My Services** tab. You can modify the catalog name and category for your service in the **Service Editor**.

### To create a service catalog and category:

1. Right-click the service in the **My Services** tab and select **Open**.  
The **Service Editor** appears.

## Service Editor

**General**

Name

Description

Mark for Publish ☒

**Classification**

Catalog	Category
Test Services	Test

Add

Remove

- Enter the service information in the **General** and **Classification** sections, as described in [Table 8, page 76](#).

Table 8. Web service information

Parameter	Description
<b>General</b>	
Name	The name of the service.
Description	A description of the service.
Mark for Publish	Specifies whether this service is ready to be published. This option is enabled by default.
<b>Classification</b>	
Catalog	The name of the catalog. Click <b>Add</b> to add a new catalog entry, then select the field in the <b>Catalog</b> column to modify it.
Category	The name of the catalog category. Click <b>Add</b> to add a new category entry, then select the field in the <b>Category</b> column to modify it.

- Save your changes.  
The new or modified catalog name and category appear in the **My Services** tab.

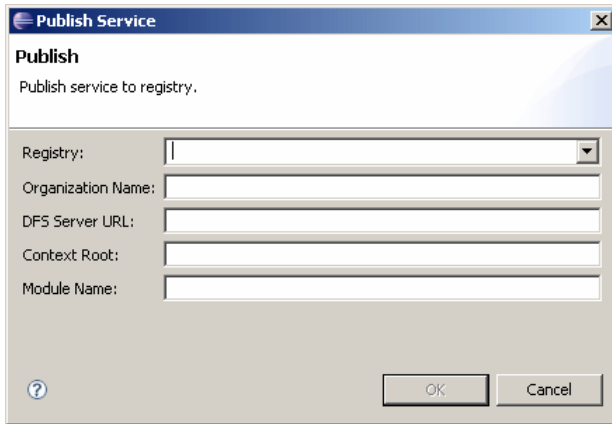
## Publishing a service

Use the **Publish Service** dialog to publish a service to a registry.

### To publish a service:

- Switch to the **Documentum Solutions** perspective and locate the service you want to publish in the **My Services** tab.

- Right-click the service and select **Publish** from the drop-down list.  
The **Publish Service** dialog appears.



- Enter the publishing information, as described in [Table 9, page 77](#), then click **OK**.

**Table 9. Publish service information**

Parameter	Description
Registry	The alias of the registry to which the service is published. Select the registry from the drop-down list.
Organization Name	The name of the organization where the service resides.
DFS Server URL	The URL of the DFS server.
Context Root	Specifies the root of the service address. For example, in the URL <code>https://127.0.0.1:7001/services/core</code> , "services" signifies the context root.
Module Name	Specifies the name of the service module. For example, in the URL <code>https://127.0.0.1:7001/services/core</code> , "core" signifies the module name.

- Click the **Service Catalogs** tab and refresh the view.  
The service appears in the **Service Catalogs** list if the service was published successfully.

## Unpublishing a service

When you unpublish a service, it is no longer available on the DFS server and does not show up in registry queries.

### To unpublish a service:

- In the **Service Catalogs** view, right-click the service and select **Unpublish**.  
The **Unpublish Services** dialog appears.
- Click **OK** to unpublish the service.  
The service does no longer appear in the **Service Catalogs** view.

## Exporting a service

When you export a service, Composer generates an archive EAR file, a JAR containing runtime resources, and an optional manifest XML file.

### To export a service:

1. Right-click the project that contains your service either in the **Documentum Navigator** or **Package Explorer** view.
2. Select **Export > Documentum > Export Service Archive**, then click Next.

The **Export Service Archive** dialog appears.

3. Enter the export service information, as described in [Table 10, page 78](#), then click **Finish**.

**Table 10. Export service information**

Parameter	Description
Archive Name	The name for the archive EAR file. The file has the format <i>&lt;archive name&gt;.ear</i> .

Parameter	Description
Context Root	Specifies the root of the service address. For example, in the URL <code>https://127.0.0.1:7001/services/core</code> , "services" signifies the context root.
Export Destination	The location on the local machine or network drive where the EAR file is saved. Click <b>Select</b> and browse for a location to store the EAR file.
Services for Export	Select the service to be exported.
Runtime Libraries	The JARs that are included in and exported with the EAR file. Click <b>Add</b> to include JAR files from your Composer workspace. Click <b>Add External JAR</b> to include JAR files from the local file system.
Runtime Resources	The runtime resources that are included in and exported with the archive EAR file. The runtime resources are packaged in a JAR file. Click <b>Select</b> and check the resource to be exported.
Generate Publish Manifest	Select this option to export a manifest XML file with the EAR file. The manifest file has the format <code>&lt;archive name&gt;-publish-manifest.xml</code> .
Organization Name	The name of the organization that created the service.

Composer creates the archive EAR file, runtime resources JAR file, and the manifest file in the export destination.

## Deploying a service

After you have exported the service and generated an EAR file, you can deploy the service by copying the EAR file to your DFS server.

### To deploy a service:

1. Generate an EAR file for the service you want to deploy, as described in [Exporting a service, page 78](#).
2. Copy the EAR file to the services directory on your DFS server.





## Managing Alias Sets

This chapter contains the following topics:

- [Alias, alias values, and alias sets, page 81](#)
- [Creating an alias set, page 81](#)

### Alias, alias values, and alias sets

Many aspects of Documentum applications involve references to specific users, groups, permission sets, and repository cabinets and folders. Instead of referencing the actual user, group, cabinet, or folder name, you can assign an alias, a symbolic name. The symbolic name is called the alias name. The actual value is called the alias value. A collection of aliases is called an alias set.

### Creating an alias set

Use the **Alias Set** editor to create an alias set.

#### To create an alias set:

1. Open the **Select a wizard** dialog in one of the following ways:
  - From the Composer menu, select **File > New > Other**.
  - In your Composer project, expand the **Artifacts** folder and right-click **Alias Set**. Select **New > Other**.The **Select a wizard** dialog appears.
2. Select **Documentum > Alias Set**, then click **Next**.  
The **New Documentum Artifact - Name and Location** dialog appears.
3. Enter the folder path and name of the project for which you want to create an alias set in the **Folder:** field, or click **Browse** to select the project from a folder list.
4. Enter a file name for the alias set in the **Artifact name:** field, then click **Finish**.  
The **Alias Set** editor appears.

**Alias Set**

**General**

Name: newaliasset

Description:

**Aliases**

Type	Name

Add Remove

Overview

5. Enter a name for the alias set in the **Name** field and an optional description in the **Description** field.
6. Click **Add** in the **Aliases** section to create one or more aliases that make up the alias set. The **New Alias** dialog appears.

**New Alias**

Please provide an alias name and type

Name:

Type: Unknown

OK Cancel

7. Enter a name for the new alias in the **Name:** field and select a type from the **Type:** drop-down list. You can create an alias for the following types:
  - Unknown
  - User
  - Group
  - User or Group
  - Cabinet Path
  - Cabinet or Folder Path
  - Permission Set
8. Click **OK** when you are finished. The **Alias Details** section appears.

9. Enter the details for the alias in the **Alias Details** section, as described in [Table 11, page 83](#).

**Table 11. Alias details**

Parameter	Description
Name	A string specifying the name of the alias.
Type	Specifies the type for which this alias is used.
Value	Specifies the parameter and value of the alias. Depending on which alias type you specified in the <b>Type</b> field of the <b>New Alias</b> dialog, different parameter and value options are available in the <b>Value</b> section. For more information about alias values, see <a href="#">Configuring alias values, page 83</a> .
Category	Alias category is a tool for developers to use to organize the aliases in their applications. Documentum software does not use this field.
Description	An optional description of the category.

10. Save your changes.

## Configuring alias values

An alias can have different parameters and values, depending on the type of alias that is specified in the **Type** field in the **Alias Details** section. [Table 12, page 84](#) describes the parameters associated with a specific alias type.

Table 12. Alias type values

Alias Type	Value Options	Description
Unknown	No value can be assigned to an unknown alias type.	
User	<ul style="list-style-type: none"> <li>• Leave It Blank</li> <li>• Parameter</li> </ul>	To assign a parameter: Click <b>Parameter</b> , then click <b>Select</b> . The <b>User Installation Parameter</b> dialog appears. Select a parameter from the listbox or click <b>New...</b> to create a user parameter.
Group	<ul style="list-style-type: none"> <li>• Leave It Blank</li> <li>• Parameter</li> <li>• Value</li> </ul>	<p>To assign a parameter: Click <b>Parameter</b>, then click <b>Select</b>. The <b>Group Installation Parameter</b> dialog appears. Select a parameter from the listbox or click <b>New...</b> to create a group parameter.</p> <p>To assign a value: Click <b>Value</b>, then click <b>Select</b>. The <b>Documentum Group Artifact</b> dialog appears. Select an artifact from the listbox, then click <b>OK</b>.</p>
User or Group	<ul style="list-style-type: none"> <li>• Leave It Blank</li> <li>• Parameter</li> </ul>	To assign a parameter: Click <b>Parameter</b> , then click <b>Select</b> . The <b>Principal (User or Group) Installation Parameter</b> dialog appears. Select a parameter from the listbox or click <b>New...</b> to create a parameter.
Cabinet Path	<ul style="list-style-type: none"> <li>• Parameter</li> <li>• Value</li> </ul>	<p>To assign a parameter: Click <b>Parameter</b>, then click <b>Select</b>. The <b>Folder Installation Parameter</b> dialog appears. Select a parameter from the listbox or click <b>New...</b> to create a parameter.</p> <p>To assign a value: Click <b>Value</b>, then click <b>Select</b>. The <b>FolderSubtype Artifact</b> dialog appears. Select an artifact from the listbox, then click <b>OK</b>.</p>
Cabinet or Folder Path	<ul style="list-style-type: none"> <li>• Parameter</li> <li>• Value</li> </ul>	<p>To assign a parameter: Click <b>Parameter</b>, then click <b>Select</b>. The <b>Folder Installation Parameter</b> dialog appears. Select a parameter from the listbox or click <b>New...</b> to create a parameter.</p> <p>To assign a value: Click <b>Value</b>, then click <b>Select</b>. The <b>FolderSubtype Artifact</b> dialog appears. Select an artifact from the listbox, then click <b>OK</b>.</p>
Permission Set	<ul style="list-style-type: none"> <li>• Parameter</li> <li>• Value</li> </ul>	<p>To assign a parameter: Click <b>Parameter</b>, then click <b>Select</b>. The <b>ACL Installation Parameter</b> dialog appears. Select a parameter from the listbox or click <b>New...</b> to create a parameter.</p> <p>To assign a value: Click <b>Value</b>, then click <b>Select</b>. The <b>Permission Set (ACL) Template Artifact</b> dialog</p>

Alias Type	Value Options	Description
		appears. Select an artifact from the listbox or click <b>New ...</b> to create an artifact.



## Managing Aspects

This chapter contains the following topics:

- [Aspect modules and aspect types, page 87](#)
- [Creating an aspect type, page 87](#)
- [Adding aspect attributes, page 90](#)
- [Configuring the aspect UI information , page 92](#)
- [Creating an aspect module, page 95](#)

## Aspect modules and aspect types

An aspect module consists of executable business logic and supporting material for an aspect, such as third-party software and documentation. An aspect customizes behavior or records metadata or both for an instance of an object type. An aspect module is comprised of the aspect type definition, the JAR files that contain the implementation classes and the interface classes for the behavior the aspect implements, and any interface classes on which the aspect depends. The module may also include Java libraries and documentation.

## Creating an aspect type

Use the **Aspect** editor to create an aspect type.

### To create an aspect type:

1. Open the **Select a wizard** dialog in one of the following ways:
  - From the Composer menu, select **File > New > Other**.
  - In your Composer project, expand the **Artifacts** folder and right-click **Types**. Select **New > Other**.

The **Select a wizard** dialog appears.

2. Select **Documentum > Aspect Type**, then click **Next**.

The **New Documentum Artifact - Name and Location** dialog appears.

- Enter the folder path and name of the project for which you want to create an aspect type in the **Folder:** field, or click **Browse** to select the project from a folder list.
- Enter a file name for the aspect type in the **Artifact name:** field, then click **Next**.

The **Aspect** editor appears with the **General** tab selected.

The screenshot shows the 'General' tab of the Aspect editor. It contains three main sections: 'Info', 'Constraints', and 'Events'. The 'Info' section has a 'Type name' field with the text 'newaspecttype'. The 'Constraints' section features a table with two columns, 'Expression' and 'Enforcement', and three buttons: 'New...', 'Remove', and 'Edit'. The 'Events' section has a table with two columns, 'Event Name' and 'Event Label', and two buttons: 'New...' and 'Remove'. At the bottom of the window, there are three tabs: 'General', 'Attributes', and 'Display', with 'General' being the active tab.

- Enter the aspect information in the **Info**, **Constraints**, and **Events** sections, as described in [Table 13, page 88](#).

**Table 13. Aspect information on General tab**

Property	Description
<b>Info</b>	
Type name	<p>A string specifying the name of the aspect. The aspect type name should be the same as the name of the aspect module that is referencing the aspect. The following rules apply to all aspect names:</p> <ul style="list-style-type: none"> <li>A maximum of 27 characters, all lower-case. The Content Server is case-insensitive and stores all type names in lower-case.</li> <li>The first character must be a letter, the remaining characters can be letters, digits, or underscores</li> <li>Cannot contain any spaces or punctuation</li> <li>Cannot end in an underscore (_)</li> </ul>
<b>Constraints</b>	
Expression	<p>Constraints are internal consistency requirements in the form of Docbasic expressions that relate the types attribute values to one another or to constant values.</p> <p>The Docbasic expression defining the constraint. Click <b>New</b> to create an expression. For more information about adding constraints, see <a href="#">Configuring constraint expressions, page 89</a>.</p>



Property	Description
Enforcement	<p>Specifies whether applications enforce this constraint or not. Click the table cell in the Enforcement column to enable or disable constraint enforcement for the associated expression. The enforcement field can have two values, as follows:</p> <ul style="list-style-type: none"> <li>• <b>disabled</b>: The constraint is disabled</li> <li>• <b>ApplicationEnforced</b>: The constraint is enforced by the applications that use this type.</li> </ul>
Events	<p>Events are specific actions on objects. You can only create and modify application events, not system events. Click <b>New</b> to enter a new event. To edit or remove an event, select the event and click <b>Edit</b> or <b>Remove</b>, respectively.</p>
Event name	<p>A string specifying the name of the event that is associated with instances of this type.</p>
Event label	<p>A string that specifies the label for the event.</p>

## Configuring constraint expressions

Constraints are internal consistency requirements in the form of Docbasic expressions that relate the aspect attribute values to one another or to constant values.

### To add a constraint expression for an aspect:

1. Click **Add** in the Constraints section of the Aspect Overview tab in the aspect editor.  
The **Edit Constraint** dialog appears.

2. Type a valid Docbasic constraint expression that resolves to true or false in the **Expression:** text box. The Docbasic expression resolves to true when the constraint is fulfilled and false when the constraint is violated.

3. Type a message for applications to display when the constraint is violated in the **Error message when constraint is violated:** text box.
4. Check the **Enforce Constraint** checkbox to instruct applications to enforce this constraint or leave it unchecked to not enforce the constraint.
5. Click **OK** to save your changes.

## Adding aspect attributes

Aspect attributes are configured in the **Attributes** tab of the aspect editor. An aspect attribute is a property that applies to all aspect of that type. When an aspect is created, its attributes are set to values that describe that instance of the aspect type. You must configure attributes for each aspect. If you create an aspect without configuring any attributes, the aspect artifact does not install correctly and causes the installation of the entire project to fail.

### To create an attribute:

1. Click the **Attributes** tab in the aspect editor to display the Attributes view.
2. Click **New** to create an attribute entry, then select the new attribute entry.

The **Aspect Attributes** view expands.

The screenshot shows the 'Aspect Attributes' dialog box. On the left, there is a list box containing 'newattribute1'. To its right are 'New' and 'Remove' buttons. The right side of the dialog is titled 'Structure' and contains the following fields: 'Name:' with the value 'newattribute1', 'Data type:' with a dropdown menu showing 'STRING', 'Length:' with the value '10', 'Repeating:' with an unchecked checkbox, and 'Non-qualifiable:' with an unchecked checkbox. Below these is a 'Default values:' section with a table that has 5 rows and 2 columns. To the right of this table are 'New' and 'Remove' buttons. At the bottom of the dialog, there are three tabs: 'General', 'Attributes', and 'Display', with 'Attributes' currently selected.

3. Configure the attribute structure, as described in [Configuring the aspect attribute structure, page 90](#).

## Configuring the aspect attribute structure

The attribute structure is configured in the **Structure** section of the **Aspect Attributes** view ([Figure 3, page 91](#)).

**Figure 3. Structure section in Aspect Attributes view**

**Structure**

Name: NewAttribute1

Data type: STRING

Length: 0

Repeating: ☐ Non-qualifiable: ☒

Default values:


New

Remove

Enter the attribute structure properties, as described in [Table 14, page 91](#).

**Table 14. Attribute structure properties**

Property	Description
Name	A string specifying the name of the new attribute. The attribute name must use all lowercase letters, cannot begin with dm_, a_, i_, r_, a numeral, space, or single quote, and cannot be named select, from, or where.
Data type	<p>The data type of the new attribute. Select one of the following data types from the drop-down list:</p> <ul style="list-style-type: none"> <li>• BOOLEAN</li> <li>• INTEGER</li> <li>• STRING</li> <li>• ID</li> <li>• TIME</li> <li>• DOUBLE</li> <li>• UNDEFINED</li> </ul>
Length	<p>This parameter only applies to attributes that use the STRING data type. Enter the number of characters for this attribute. The maximum number of characters that you can assign to this attribute depends on the database where you are installing the application.</p>
Repeating	<p>Specifies whether this attribute can have more than one value. Select the checkbox to allow more than one value for this attribute.</p>

Property	Description
Non-qualifiable	Specifies whether the attribute is qualifiable or non-qualifiable.  The properties and values of a non-qualifiable attribute are stored in a serialized format and do not have their own columns in the underlying database tables that represent the object types for which they are defined. Consequently, non-qualifiable attributes cannot be used in queries because they are not exposed in the database.
Default values	Lets you specify one default value for a single-value attribute or multiple default values for a repeating attribute.

## Configuring the aspect UI information

The **Aspect UI Information** view lets you specify which aspect attributes are displayed in Documentum clients and custom applications. Ensure that the client application that you are using supports displaying attribute information for aspects.

**Note:** Webtop does not support displaying attributes for aspects.

**Figure 4. Aspect UI information view**

### To configure one or more attributes to be displayed in clients:

1. Click the **Display** tab in the aspect editor to display the **Aspect UI Information** view (Figure 4, page 92).
2. Enter the aspect UI information as described in Table 15, page 93.

Table 15. Aspect UI information

Property	Description
<b>Display Configuration</b>	
Scope	The name of the application, in which the aspect is displayed. The name of the application must exist in the repository.  <b>Note:</b> Webtop does not support displaying attributes for aspects.
Display configuration list	Specifies the tab on which the aspect attribute is displayed. You can add, remove, rename, and change the position of a tab, as follows: <ul style="list-style-type: none"> <li>Click <b>New</b> to add a new tab. The <b>Display Configuration</b> dialog appears. For more information about adding a tab to display an attribute in a client application, see <a href="#">Adding a tab</a> , page 93.</li> <li>To remove a tab, select the tab name in the list, then click <b>Remove</b>.</li> <li>To rename a tab, select the tab name in the list, then click <b>Rename</b>.</li> <li>To change the order in which the tabs are displayed, select the tab name in the list, then click <b>Up</b> or <b>Down</b> to move the tab to the desired position.</li> </ul>
Attributes in display configuration	Lets you modify the attributes that are displayed on a tab.
<b>Application Interface UI</b>	
Type label	A string that the client application displays for this aspect.
User help	Optional description for the aspect that is displayed in the application.
Comments for developers	Optional comments for developers.

## Adding a tab

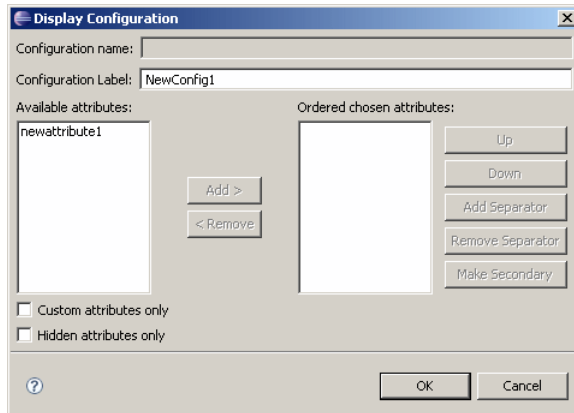
Use the **Display Configuration** dialog to add a tab to display an attribute.

### To add a tab to display an attribute:

1. Click **New** in the **Display** configuration list section of the **Aspect UI Information** view ([Figure 13, page 192](#)). A default tab (**NewConfig1**) for the new tab appears in the **Display** configuration list box.

2. Select the default tab and then click **Edit**.

The **Display Configuration** dialog appears.



3. Configure the tab properties, as described in [Table 16, page 94](#).

**Table 16. Tab configuration properties**

Tab properties	Description
Configuration name	A string that specifies the internal object name of the tab. You cannot change this name in this dialog.
Configuration label	A string that specifies the name that is displayed on the tab in the client application.
Available attributes	Shows a list of the attributes that can be displayed on the tab. Select the attribute that you want to display on the tab and click <b>Add</b> . The attribute appears in the Ordered chosen attributes list. If the available attributes list is empty, no attributes have been configured yet. For more information about configuring attributes, see <a href="#">Adding aspect attributes, page 90</a> .
Ordered chosen attributes	Specifies which attributes are displayed on the tab and how they are displayed. You can arrange how the attributes are displayed on the tab by selecting the attribute and using the following buttons: <ul style="list-style-type: none"> <li>• <b>Up</b>: Moves the attribute up in the display order.</li> <li>• <b>Down</b>: Move the attribute down in the display order.</li> <li>• <b>Add Separator</b>: Adds a separator between the selected and the following attribute.</li> <li>• <b>Remove Separator</b>: Removes the separator.</li> <li>• <b>Make Secondary</b>: Force attributes to be displayed on a secondary page, if not all attributes can fit on one tab.</li> </ul>

Tab properties	Description
Custom attributes only	Select this option to display only custom attributes in the Available attributes list.
Hidden attributes only	Select this option to display only hidden attributes in the Available attributes list.

- Click **OK** to save your changes.

## Creating an aspect module

Before you can create an aspect module, create a Documentum project. For more information about creating a Documentum project, see [Creating a project, page 22](#).

### To create an aspect module:

- Open the Select a wizard dialog in one of the following ways:
  - From the Composer menu, select **File > New > Other**.
  - In your Composer project, expand the **Artifacts** folder and right-click **module**. Select **New > Other**.

The **Select a wizard** dialog appears.

- Double-click the Documentum folder to expand it, then select **Aspect Module**.  
The **New Documentum Artifact Name and Location** dialog appears.
- Enter a name for the new aspect module, then click **Finish**.

**Note:** Use the same name for the aspect module and the aspect type associated with the module.

The **Aspect Module** editor appears with the **General** tab selected.

**General**

**Info**  
Specify the module name and type  
Name:   
Type:

**Description**  
Specify the module author and enter a description for the module  
Author:   
Description:

**Required Modules**  
Specify other modules that this module requires  
  
Add...  
Remove  
Edit...

**Javadoc**  
Specify Javadoc and additional resources that can be downloaded at deployment time  
Javadoc:  Select...

**Core JARs**  
Select the implementation and interface JARs for the application. Implementation JARs typically contain the classes that implement the interfaces in the interface JARs. However, implementation JARs may contain both classes and interfaces. A class functions as the entry point to the module.  
Implementation JARs:  Add... Remove Edit...  
Class name:  Select...  
Interface JARs:  Add... Remove Edit...

General | Deployment | Runtime | Aspect Type

- Enter the required and optional properties in the Info, Description, Required Modules, Javadoc, and Core JARs sections, as described in [Table 17, page 96](#).

**Table 17. Properties in module editor General tab**

Property	Description
<b>General</b>	
Name	A string specifying the name of the module. Required parameter. The name can have up to 255 characters.
Type	A string specifying the type of the module. Required parameter. An aspect module can only be of the type Aspect.
<b>Description</b>	
Author	Contact information for the module author. Optional parameter
Description	Description for the module, not exceeding 255 characters. Optional parameter.
<b>Required Modules</b>	
	Specifies modules that this module requires to function properly. Click <b>Add</b> to open the Module Artifact dialog. Select a module from the listbox and click <b>OK</b> , or click <b>New</b> to create a module.



Property	Description
<b>Javadoc</b>	Specifies Javadocs and other resources that can be downloaded with the aspect module at runtime. Click <b>Select</b> to open the SysObject Subtype Artifact dialog. Select a SysObject that contains the Javadoc or resource content from the list or click <b>New</b> to create a SysObject containing the content to be downloaded.
<b>Core JARs</b>	
Implementation JARs	Implementation of the module. Required parameter. Click <b>Add</b> to add implementation JARs from your local machine.
Class name	Primary Java implementation class for the module. Required parameter. TBOs must implement the IDfBusinessObject interface; SBOs must implement the IDfService interface; and all modules must implement the IDfModule interface.
Interface JARs	Java interfaces that this module implements. Optional parameter. Click <b>Add</b> to add interface JARs from your local machine.

- Click the **Deployment** tab and configure the module dependencies as described in [Configuring aspect module deployment](#) , page 97.
- Click the **Runtime** tab to configure the runtime environment for this module, as described in [Configuring the aspect module runtime environment](#), page 99.
- Click the **Aspect Type** tab to configure an aspect type for this module, as described in [Configuring the aspect type](#), page 100.

## Configuring aspect module deployment

The **Deployment** tab lets you link Java libraries and other modules to the module you are creating or editing.

### To configure module deployment:

- Click the Deployment tab in the aspect module editor.  
The **Deployment** view appears.

**Deployment**

**Java Libraries**  
Specify additional Java libraries used by the module

Add...  
Remove  
Edit...

**Attachments**  
Additional objects

Downloadable	Name

Add...  
Remove

**Logging**  
Enter a message that is logged on the client at the specified log level after the module has been downloaded

Post-download message:

Log level: WARN

General | Deployment | Runtime | Aspect Type

- In the **Java Libraries** section, click **Add** to add Java libraries for this module. The **Jar Def Java Library Artifact** dialog appears.

**Jar Def Java Library Artifact**

Type to select artifact (? = any character, \* = any string): New...

Matching Artifacts:

- MyJavaLibrary

Properties

OK Cancel

Select a Java library from the listbox and click **OK** or click **New** to create a Java Library. You can only link existing Java libraries into this module. You cannot modify an existing library that is shared by multiple modules. For more information about creating a Java Library, see [Linking and configuring a Java Library, page 109](#).

- In the **Attachments** section, specify additional objects that are available for download when the module is deployed.
- In the **Logging** section, specify a post-download message and select a log level for the message. The log level can have the following values:
  - WARN**: The post-download message is logged as a warning.
  - NONE**: The post-download message is not logged.
  - INFO**: The post-download message is logged as an informational message.
  - DEBUG**: The post-download message is logged at debug level.
- Save your changes.

# Configuring the aspect module runtime environment

The runtime environment lets you configure optional properties that are executed at runtime, such as version requirements, Java system properties, statically deployed classes, and local resources.

## To configure the runtime environment:

1. Click the **Runtime** tab in the aspect module editor.

The **Runtime** view appears.

2. Specify the version requirements, Java system properties, statically deployed classes, and local resources, as described in [Table 18, page 99](#).

**Table 18. Module runtime environment properties**

Property	Description
<b>Version Requirements</b>	This section lets you specify the DFC and Java VM versions required on the client for the module to function properly.
Min DFC version	The minimum DFC version on the client machine for this module to work properly.
Min VM version	The minimum Java VM version on the client machine for this module to work properly.
<b>Java System Properties</b>	This section lets you specify Java system properties as name-value pairs. When the module is downloaded, the client machine is checked to see if all the specified Java properties match the properties on the client machine. Click <b>Add</b> to enter placeholders for the name and value of the Java system property, then click the <b>Name</b> and the <b>Value</b> fields to modify the property name and value.
Name	Name of the Java system property.

Property	Description
Value	Corresponding value for the Java system property name.
<b>Statically Deployed Classes</b>	This section lets you specify static Java classes that are required for the module to function properly. When the module is downloaded, the class path is checked for the specified Java classes.
Fully qualified class name	Fully qualified Java class names. Enter the class name and click <b>Add</b> .
<b>Local Resources</b>	This section lets you specify files that are required on the local machine for the module to function properly. When the module is downloaded, the client machine is checked for the specified files specified.
File path relative to deployment	Full file path. Enter the file name and path and click <b>Add</b> .

3. Save your changes.

## Configuring the aspect type

Use the **Aspect Type** tab to configure the aspect type.

### To configure the aspect type:

1. Click the **Aspect Type** tab in the aspect module editor.  
The **Aspect Type** view appears.

The screenshot shows the 'Aspect Type' configuration window. It features a tabbed interface with the 'Info' tab active. The 'Info' section includes a 'Type reference' field with a 'Select...' button, a 'Copy Aspect when' section with two checked options ('Object is copied' and 'Object is versioned'), an 'Aspect category' section with a text input, an 'Add' button, and a list box with a 'Remove' button, and a 'Target object type' section with a text input, an 'Add...' button, and 'Remove' and 'Edit...' buttons. The bottom of the window shows a series of tabs: 'General', 'Deployment', 'Runtime', and 'Aspect Type'.

2. Configure the aspect type properties in the **Info** section, as described in [Table 19, page 101](#).

Table 19. Aspect type properties

Property	Description
<b>Info</b>	
Type reference	<p>Specifies the aspect type that is associated with this aspect module.</p> <p>Click <b>Select</b> to add a type reference. The <b>Select Aspect Artifact</b> dialog displays. Select an aspect type from the list or click <b>New</b> to create an aspect type. For more information about creating an aspect type, see <a href="#">Creating an aspect type, page 87</a>.</p>
Copy aspect	Specifies whether the aspect is copied with the associated object if the object is copied. By default, the aspect is copied with the object.
Version aspect	Specifies whether the aspect is copied with the associated object if the object is versioned. By default, the aspect is copied with the object.
Aspect category	<p>Specifies the aspect category that is associated with this aspect module.</p> <p>To add an aspect category, select an aspect category from the list and click <b>Add</b>.</p>
Target object type	<p>Specifies to which object types the aspect can be attached.</p> <p>Click <b>Add</b> to add a target object type. The <b>Select Type Artifact</b> dialog displays. Select a type from the list or click <b>New</b> to create a type. For information about creating types, see <a href="#">Chapter 18, Managing Types</a>.</p>



# Managing Formats

This chapter contains the following topics:

- [Formats](#), page 103
- [Creating a format artifact](#), page 103

## Formats

A format object contains information about a file format recognized by Content Server. A predefined set of file formats is installed by default when a repository is configured.

## Creating a format artifact

Use the format editor to create a format artifact.

### To create a format artifact:

1. Open the **Select a wizard** dialog in one of the following ways:
  - From the Composer menu, select **File > New > Other**.
  - In your Composer project, expand the **Artifacts** folder and right-click **Formats**. Select **New > Other**.

The **Select a wizard** dialog appears.

2. Select **Documentum > Format**, then click **Next**.

The **New Documentum Artifact - Name and Location** dialog appears.

3. Enter a folder path and a name for the format artifact, or accept the default path and name, then click **Finish**.

The **Format** editor appears.

The screenshot shows a 'Format Properties' dialog box with three main sections:

- General:** Includes fields for Name (MyFormat), Description, Default File Extension, Com Class ID, MIME Type, Macintosh Creator, Macintosh Type, and Is Hidden (unchecked). There is a 'Classes' list with 'Add', 'Remove', 'Up', and 'Down' buttons.
- Digital Asset Management:** Includes fields for Asset Class, Default Storage (with a 'Select...' button), Filename Modifier, and a checkbox for Rich Media Enabled (unchecked).
- Full Text Index:** Includes a checkbox for 'Can be Indexed' (checked), a radio button for 'Index with Filter' (selected) with a dropdown menu set to 'Universal', and a radio button for 'Index with Rendition' with a 'Select...' button.

An 'Overview' tab is visible at the bottom left.

4. Enter the format properties in the **General**, **Digital Asset Management**, and **Full Text Index** sections, as described in [Table 20, page 104](#).

**Table 20. Format artifact properties**

Parameter	Description
<b>General</b>	
Name	An ASCII string specifying the name of the format. The string cannot be longer than 64 characters.
Description	A string describing the format. The string cannot be longer than 64 characters.
Default file extension	A string specifying the DOS extension that is used when a file with this format is copied into the common area, client local area, or storage. The string cannot be longer than 10 characters.
COM class ID	A string specifying the class ID recognized by the Windows registry for a content type. The string cannot be longer than 38 characters.
MIME type	A string specifying the Multimedia Internet Mail Extension (MIME) for the content type. The string cannot be longer than 64 characters.
Macintosh creator	A string with up to 4 characters used internally for managing Macintosh resource files.
Macintosh type	A string with up to 4 characters used internally for managing Macintosh resource files.
Is hidden	Used by client applications to determine whether to display this format object in the client application's user interface. If selected, the format is not displayed in the client's user interface.
Classes	Specifies the class or classes of formats to which the format belongs. For example, the xml, xsd, and xsl formats belong to the XML and MSOffice classes.

### Digital Asset Management



Parameter	Description
Asset class	A string with up to 32 characters that applications use to classify the asset type (for example, audio, video, image) of the contents of objects with this format.
Default storage	Specifies the default storage area (identified by its object_id) where the contents of the objects with this format are stored. If a storage type is not specified for a SysObject, the default storage for the associated format is used. If no default storage is specified, then the storage type specified for the object type is used. If none of these are specified, then turbo storage is used as the default storage.
File name modifier	Specifies a string that a client application can append to a file name when multiple renditions (of an object) having the same extension are exported. For example, if you specify "_th" as the filename_modifier for the jpeg_th format, then when a rendition, my_picture.jpeg with a jpeg_th format, is exported, the rendition's file name is my_picture_th.jpeg.
Rich media enabled	Indicates whether Content Server automatically generates thumbnails, auto proxy and metadata for its contents.
<b>Full Text Index</b>	
Can be indexed	Indicates whether an object's content with the format can be full-text indexed.
Index with filter	Name of the Verity topic filter to use for full-text indexing. The topic filter can have the following values: <ul style="list-style-type: none"> <li>• Universal</li> <li>• None</li> </ul>
Index with rendition	A string specifying the format to which this format must be transformed for full-text indexing. Click <b>Select</b> to select a format from the listbox.



# Managing JARs and Java Libraries

This chapter contains the following topics:

- [JAR definitions, JARs and Java libraries, page 107](#)
- [Creating a JAR Definition, page 107](#)
- [Linking and configuring a Java Library, page 109](#)

## JAR definitions, JARs and Java libraries

A Java ARchive or JAR file is an archive file that aggregates many files into one. It is used to distribute Java classes and associated metadata, and can serve as building block for applications and extensions. The JAR files themselves can be bundled in a Java library.

There are two types of JAR files, interface JARs and implementation JARs. Interface JARs contain Java interface classes and the implementation JARs contain the classes that implement the interface classes. Generally, the interface classes and the implementation classes are aggregated in separate JAR files, however in some cases a JAR file can contain both, interface and implementation classes.

Composer lets you create a definition that points to the JAR files and Java libraries. The definition encapsulates the JAR files and Java libraries and links them to artifacts, such as modules.

## Creating a JAR Definition

Composer lets you create a JAR definition that points to JAR files containing Java interface and implementation classes. You can create a JAR definition either from the module editor or from the Composer main menu.

### To create a JAR definition:

1. Open the Select a wizard dialog in one of the following ways:
  - From the Composer menu, select **File > New > Other**.
  - In your Composer project, expand the **Artifacts** folder and right-click **JAR Definitions**. Select **New > Other**.

The **Select a wizard** dialog appears.

2. Select **Documentum > JAR Definition**, then click **Next**.

The **New Documentum Artifact - Name and Location** dialog appears.

3. Enter a folder path and a name for the JAR definition, or accept the default path and name, then click **Finish**.

The **JAR** definition editor appears.

4. Enter the JAR definition properties and add file content, as described in [Table 21, page 108](#).

**Table 21. JAR definition properties**

Property	Description
<b>Info</b>	
Name	A string specifying the name of the JAR definition.
Min VM version	A string with a maximum of 32 characters specifying the Java VM version level required by this JAR definition.
Type	<p>Specifies the type of JAR definition. The type can have the following values:</p> <ul style="list-style-type: none"> <li>• <b>Implementation:</b> The JAR file definition contains implementation classes or implementation and interface classes.</li> <li>• <b>Interface:</b> The JAR file definition contains only interface classes.</li> <li>• <b>Interface and Implementation:</b> This JAR file definition points to both, interface and implementation classes.</li> </ul>

Property	Description
<b>JAR Content</b>	<p>Specifies the local files that are aggregated in the JAR file.</p> <p>Click <b>Browse</b> to select a JAR file on the local machine. The <b>Select content from a JAR file</b> dialog appears. Select the JAR file from the list and click <b>Open</b>.</p> <p>To view the content of a selected JAR file, click the <b>File</b> link, then select the editor you want to use to view the content.</p>

5. Save your changes.

## Linking and configuring a Java Library

Composer lets you link Java libraries to a module. A Java library contains interface and implementation JARs for the module. You can link a Java library either from the module editor or from the Composer main menu.

### To link a Java library:

1. Open the **Select a wizard** dialog in one of the following ways:
  - From the Composer menu, select **File > New > Other**.
  - In your Composer project, expand the **Artifacts** folder and right-click **Java Library**. Select **New > Other**.

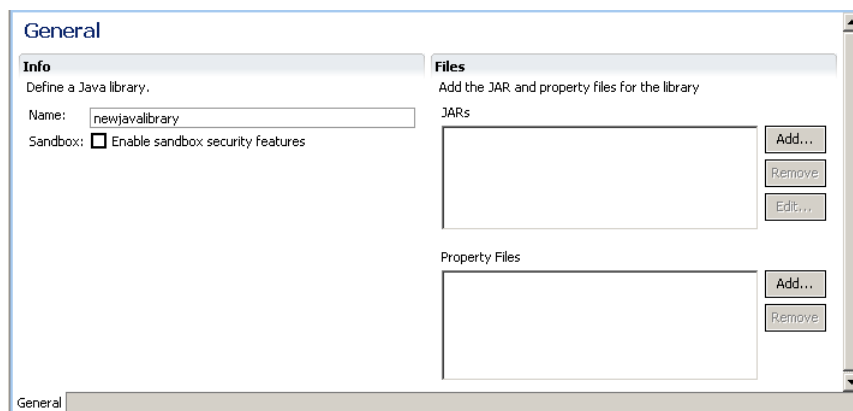
The **Select a wizard** dialog appears.

2. Select **Documentum > Java Library**, then click **Next**.

The **New Documentum Artifact - Name and Location** dialog appears.

3. Enter a folder path and a name for the Java library, or accept the default path and name, then click **Finish**.

The **Java Library** editor appears.



4. Configure the Java library, as described in [Table 22, page 110](#).

**Table 22. Java library properties**

Property	Description
<b>Info</b>	
Name	A string specifying the name of the Java library.
Sandbox	Specifies whether this Java library uses a sandbox. If checked, the Java library uses a sandbox. By default, the sandbox is disabled.
<b>Files</b>	
	The JARs and property files to be included in the Java library. Click <b>Add</b> and select the JARs and properties files from the listbox.

5. Save your changes.

## Managing Lifecycles

This chapter contains the following topics:

- [Lifecycles, page 111](#)
- [Creating a lifecycle, page 112](#)
- [Configuring lifecycle properties, page 113](#)
- [Adding and configuring lifecycle states, page 115](#)
- [Configuring state entry criteria, page 118](#)
- [Configuring state actions, page 120](#)
- [Configuring post-change information, page 131](#)
- [Configuring state attributes, page 131](#)
- [Deleting a state, page 132](#)
- [Deleting a lifecycle, page 133](#)

## Lifecycles

A lifecycle specifies business rules for changes in the properties of an object, such as a document. In other words, a lifecycle defines the different stages of an attached document. For example, a document could be in a draft state, a review stage, and a finalized state. A lifecycle does not define what activities happen to the document while it resides in a state. Activities are defined by workflows.

From a high-level view, a lifecycle consists of an attached object and various states that define the properties of the object. Planning a lifecycle includes determining the following:

- Object type(s) that can be attached to the lifecycle.
- Normal states, including entry and exit criteria, state actions, and procedures.
- Exception states, including entry and exit criteria, state actions, and procedures.
- Validation procedures.
- Alias sets.

## Lifecycle object types

In general, any content object type can be attached to a lifecycle. SysObjects are the supertype, directly or indirectly, of all object types that can have content, and any SysObject and SysObject subtype can be attached to a lifecycle. The SysObject subtype most commonly associated with content is dm\_document.

A lifecycle requires a primary object type and can include secondary object types. The primary object type specifies the type of document that can be attached to the lifecycle. A document can only be attached to the lifecycle, if the document type matches the primary object type. The primary object type can only be dm\_sysobject or one of its subtypes. Secondary object types are subtypes of the primary object type.

By default, the Composer provides the dm\_sysobject and its subtypes in the lifecycle editor.

You can also create an object type. For more information about object types and creating an object type, see [Chapter 18, Managing Types](#).

## Creating a lifecycle

You can create a lifecycle using the **Lifecycle** editor in Composer.

### To create a lifecycle:

1. Open the lifecycle editor in one of the following two ways:
  - From the Composer main menu, select **File > New > Other**.
  - In your Composer project, expand the **Artifacts** folder and right-click **Lifecycles**. Select **New > Other**.

The **Select a wizard** dialog appears.

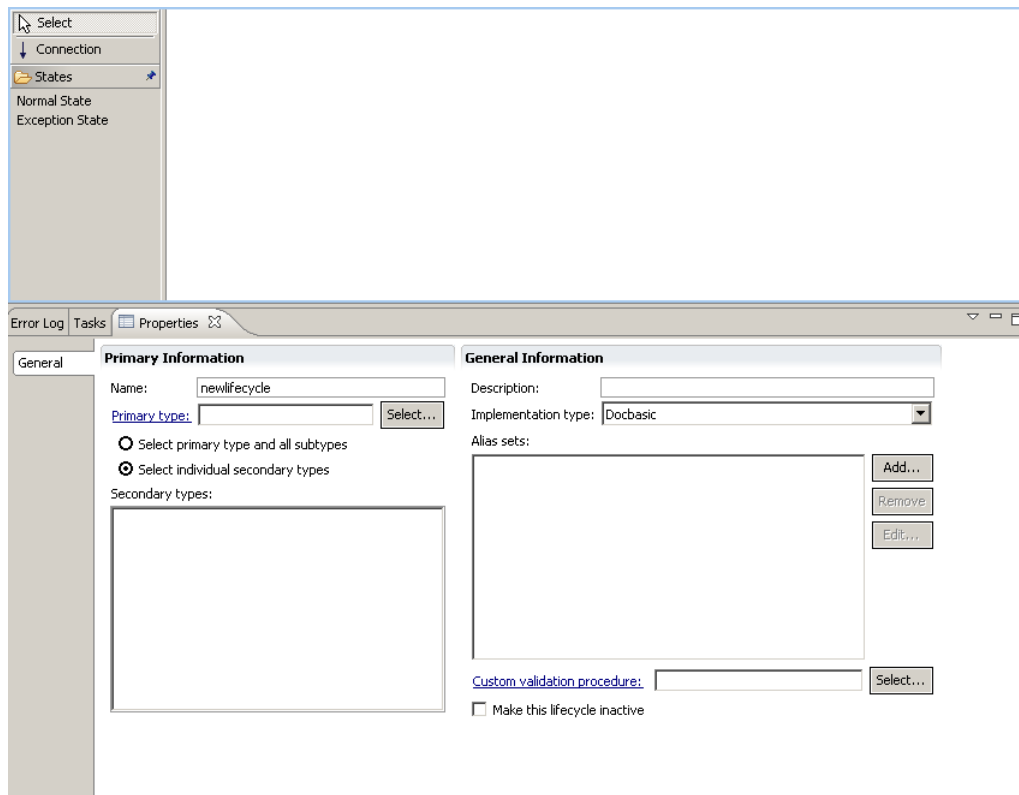
2. Select **Documentum Artifact > Lifecycle**, then click **Next**.

The **New Documentum Artifact – Name and Location** dialog appears.

3. Enter the full path name of the folder in which you want to create the lifecycle in the **Folder:** field or click **Browse** to select a folder from drop-down list.
4. Enter the name of the lifecycle in the **Artifact name:** field, or accept the default name.
5. Click **Finish**.

The **Lifecycle** editor appears.





6. Configure the properties of the lifecycle, as described in [Configuring lifecycle properties, page 113](#).
7. Configure lifecycle states, as described in [Adding and configuring lifecycle states, page 115](#).
8. Save your changes.

## Configuring lifecycle properties

The **General** tab of the lifecycle properties page ([Figure 5, page 114](#)) lets you configure the properties of a lifecycle, such as the primary object type, secondary object types, a validation procedure, implementation type, and alias sets.

Figure 5. Lifecycle properties tab

The screenshot shows a 'Properties' window with a 'General' tab. It is divided into two main sections: 'Primary Information' and 'General Information'.

- Primary Information:**
  - Name:** ChangeNotice
  - Primary type:** dcm\_change\_notice (with a 'Select...' button)
  - Radio buttons:
    - ☐ Select primary type and all subtypes
    - ☒ Select individual secondary types
  - Secondary types:** (An empty list box)
- General Information:**
  - Description:** (Empty text field)
  - Implementation type:** Docbasic (dropdown menu)
  - Alias sets:** dcm52 (list box with 'Add...', 'Remove', and 'Edit...' buttons)
  - Custom validation procedure:** (Empty text field with a 'Select...' button)
  - ☐ Make this lifecycle inactive

Configure the **Primary Information** and **General Information** for the lifecycle, as described in [Table 23, page 114](#).

Table 23. Lifecycle properties tab parameters

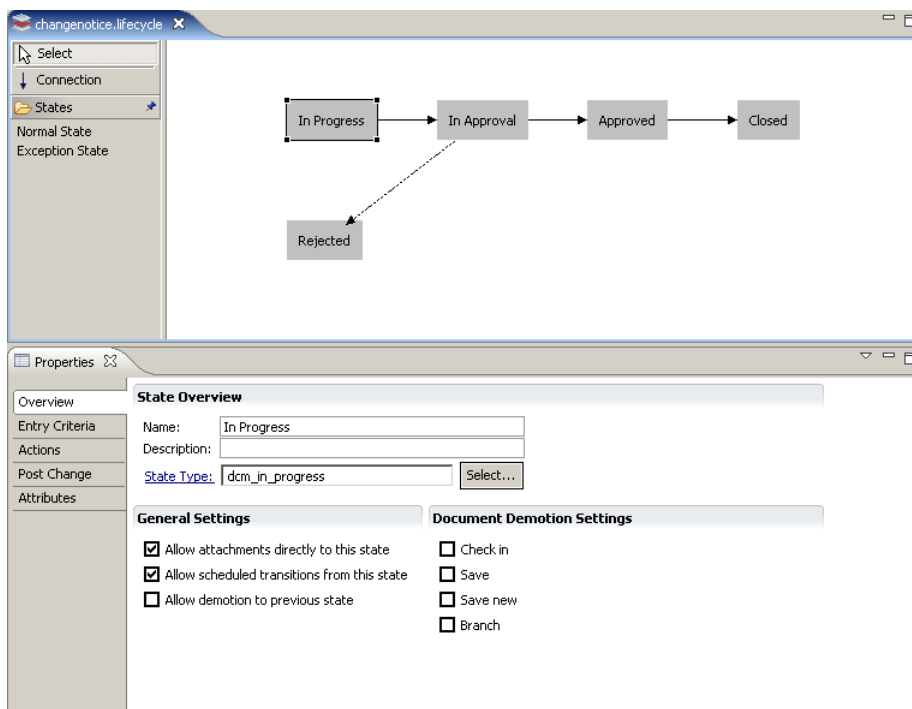
Parameter	Description
<b>Primary Information</b>	
Name	A string specifying the name of the lifecycle.
Primary type	Assign a primary object type, using one of the following methods: <ul style="list-style-type: none"> <li>Click <b>Select</b>. The <b>Select Type Artifact</b> dialog appears. Select the primary object type from the listbox.</li> <li>Click the <u>P</u>imary type link to create an object type. The <b>New Object Type Artifact</b> wizard appears. For more information about creating a custom object type, see <a href="#">Creating a standard object type, page 176</a>.</li> </ul>
Select primary type and all subtypes	If selected the lifecycle applies to the primary type and all its subtypes.
Select individual secondary types	Select this option if you want to assign individual secondary types. Choose the secondary types from the Secondary types listbox.

Parameter	Description
Secondary types	Displays the secondary types that can be assigned to this lifecycle. Secondary object types are subtypes of the primary object type that is specified in the <b>Primary type</b> field. If you specify secondary object types, only objects of the secondary object type can be attached to the lifecycle. If you do not select any secondary object types, the attached type must match the primary object type.
<b>General Information</b>	
Implementation type	Specifies the implementation type. Select an implementation type from the drop-down list. There are two implementation types, as follows: <ul style="list-style-type: none"> <li>• Docbasic</li> <li>• Java</li> </ul>
Alias sets	Specifies the alias set associated with this lifecycle. Add one or more alias sets by clicking <b>Add</b> next to the Alias sets field in the <b>General Information</b> section. The aliases are resolved to real user names or group names or folder paths when the lifecycle executes. For more information about alias sets, see <a href="#">Chapter 6, Managing Alias Sets</a> .
Custom validation procedure	Specifies the validation procedure associated with this lifecycle. Assign a validation procedure using the <b>Validation Procedure</b> field. Click <b>Browse</b> to select a validation procedure.
Make this lifecycle inactive	Select this option to deactivate this lifecycle.

## Adding and configuring lifecycle states

Once you have created the lifecycle, you can add states. Lifecycle states are added in the main window of the lifecycle editor in form of a state diagram ([Figure 6, page 116](#)). Each state appears as a rectangle, arrows designate transitions from one state to the next.

Figure 6. Lifecycle editor with state diagram



There are two types of lifecycle states, as follows:

- Normal state

Normal states follow a linear progression, from the first (base) state to the last (terminal) state.

- Exception state

Exception states handle any deviation from the linear progression. Each normal state has either zero or one exception state into which documents can transition from a normal state. An exception state can serve more than one normal state.

### To add a lifecycle state:

1. In the lifecycle editor palette, click **Normal State** or **Exception State**, depending on the type of state you want to add.
2. Move your mouse cursor over the editor window and left-click inside the editor window to draw the lifecycle state.
3. Enter the state properties in State Overview, General Settings, and Document Demotion Settings in the Overview tab, as described in [Table 24, page 116](#).

**Table 24. State properties in Overview tab**

Properties	Description
<b>State Overview</b>	
Name	A string that specifies the name of the lifecycle state.

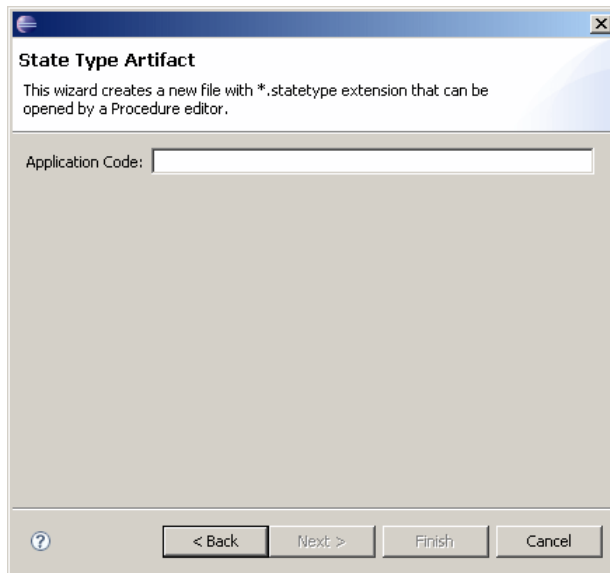
Properties	Description
Description	A string that describes the purpose or function of the lifecycle state. Optional parameter.
State Type	<p>A string that specifies the state by type. A state type identifies a state that a client application uses. Certain client applications identify a state by type instead of the name.</p> <p>Type an existing state type in the <b>State type</b> field, click <b>Select</b> and select a state type from the list, or click <b>State Type</b> to create a state type. For more information about creating state types, see <a href="#">Creating a state type</a> , page 118.</p>
<b>General Settings</b>	
Allow attachments directly to this state	If this setting is checked, users can attach a document to this state. The document type must match the primary and secondary object types that were specified for the lifecycle.
Allow scheduled transitions from this state	If this setting is checked, a document can transition from this state to the next state as the result of a time schedule, and no explicit user action is required.
Allow demotions to previous state	If this setting is checked, a document attached to this state can be moved back to the previous state or the base (first) state.
<b>Document Demotion Settings</b>	
Check in	If this setting is checked, a document is automatically returned to the base state when the document is checked in. The document must pass the base state's entry criteria for the check in to succeed.
Save	If this setting is checked, a document is automatically returned to the base state when the document is saved. The document must pass the base state's entry criteria to be saved successfully.
Save new	If this setting is checked, a document is automatically returned to the base state when the document is saved as a new document. The document must pass the base state's entry criteria to be saved successfully.
Branch	If this setting is checked, a document is automatically returned to the base state when the document is branched.

## Creating a state type

A state type identifies a state that a client application uses. Certain client applications identify a state by type instead of the name. You can create a state type artifact from the lifecycle state editor.

### To create a state type:

1. In the **Lifecycle** editor, select the lifecycle state for which you want to create a state type.
2. In the **State Overview** section, click the **State Type** link.  
The **New Documentum Artifact – Name and Location** dialog appears.
3. Enter the full path name of the folder in which you want to create the state type in the **Folder:** field or click **Browse** to select a folder from drop-down list.
4. Enter a name for the state type in the **Artifact name:** field, or accept the default name. The state type name must be unique.
5. Click **Next**. The **State Type Artifact** dialog appears.



6. Enter the application code. Application code is a string that the client application recognizes, for example, "dm\_dcm" for Documentum Compliance Manager (DCM). See the client application's documentation for information about its application code and valid state types.
7. Click **Finish**.

## Configuring state entry criteria

State entry criteria specify the conditions a document must meet to enter a stage and a procedure that is executed when the document enters the stage. State entry criteria are configured in the **Entry Criteria** tab on the **Properties** page.

## To configure state entry criteria:

1. Click the **Entry Criteria** tab on the Properties pane to display the State Entry Criteria page.

2. In the **State Entry Criteria** section click **Add** to add one or more state entry criteria rules. Click any field in a row to enter or edit a value, as described in [Table 25, page 119](#).

**Table 25. State entry criteria**

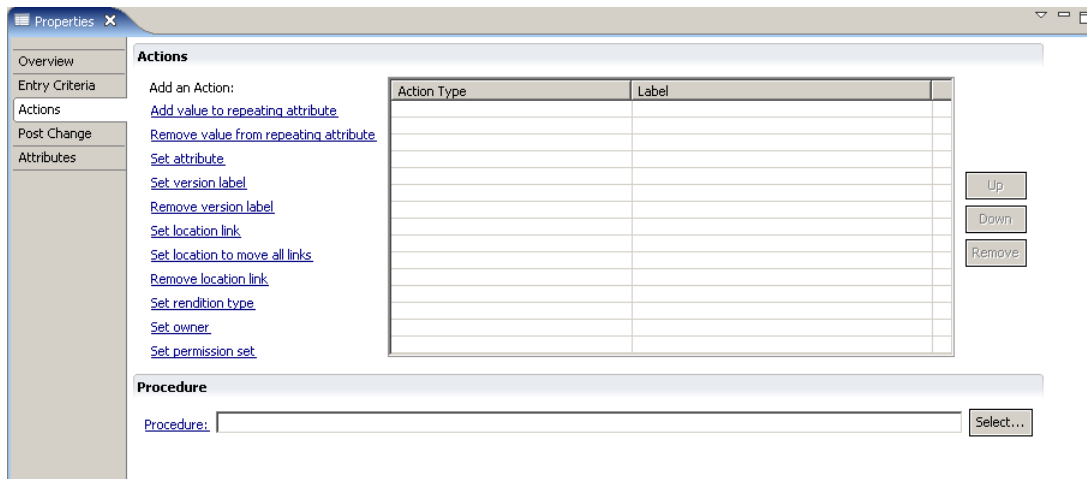
Property	Description
<b>State Entry Criteria</b>	
Logic operator	A logic operator that can have the values AND and OR.
Attribute name	The name of the attribute that specifies the entry criteria for the state. Click the field and use the drop-down list to select an attribute name. The list only shows the attributes that are valid for the primary object type associated with the lifecycle.
Index	Specifies the position of the entry criteria in the entry criteria list. By default, the index is set to NONE.
Relational operator	A relational operator that can have the following values: <ul style="list-style-type: none"> <li>• &gt; (greater than)</li> <li>• &gt;= (greater or equal)</li> <li>• &lt; (less than)</li> <li>• &lt;= (less or equal)</li> <li>• = (equal)</li> </ul>
Attribute value	A string that specifies the value of the attribute.

Property	Description
<b>Procedure</b>	
Procedure	<p>The procedure that is executed when a document enters this stage of the lifecycle. Enter a procedure name in one of the following ways:</p> <ul style="list-style-type: none"> <li>Click <b>Select</b>. The <b>Procedure Artifact</b> dialog appears. Select a procedure from the listbox or click <b>New ...</b> to create a procedure.</li> <li>Click the <b><u>Procedure:</u></b> link to create a procedure.</li> </ul> <p>For more information about creating a procedure, see <a href="#">Creating a procedure, page 157</a>.</p>

## Configuring state actions

For each state, you can define actions to be performed on an object entering the state. The actions on entry are performed after the entry criteria are evaluated. The actions must complete successfully before the object can enter the state. All state actions can be configured in the **Actions** tab of the **Properties** page.

Figure 7. Lifecycle state actions



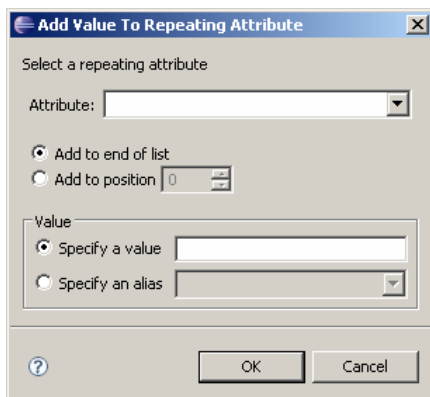
## Configuring repeating attributes

Repeating attributes are attributes that can have more than one value. For example, the document object's authors attribute is a repeating attribute since a document can have more than one author.



### To configure repeating attributes:

1. In the lifecycle state diagram, click the state for which you want to configure one or more repeating attributes.
2. Click the **Actions** tab in the **Properties** pane.  
The **Actions** page appears.
3. Click the **Add value to repeating attribute** link.  
The **Add Value to Repeating Attribute** dialog appears.



4. Enter the name of the repeating attribute, the position, and the value, as described in [Table 26](#), [page 121](#).

**Table 26. Add repeating attribute properties**

Property	Description
Attribute	The name of the repeating attribute. You can either type the name in the <b>Attribute</b> field or select a name from the drop-down list. The list only displays the attributes that are valid for the primary object you assigned to the lifecycle.
Add to end of list	Stores the position of the attribute at the end of the repeating attribute list. All repeating attributes are stored in a list, therefore an index value must be stored with the attribute's name.
Add to position	Stores the position of the attribute at the specified index position in the repeating attribute list. All repeating attributes are stored in a list, therefore an index value must be stored with the attribute's name.
Value	The value of the repeating attribute.

Property	Description
Specify a value	Select this option if you want the repeating attribute to be stored as a value, and enter the value.
Specify an alias	Select this option if you want the repeating attribute to be stored as an alias, then select the alias from the drop-down list. If the drop-down list does not show any aliases, no aliases have been configured for the project yet.
For more information about configuring aliases, see <a href="#">Chapter 6, Managing Alias Sets</a> .	

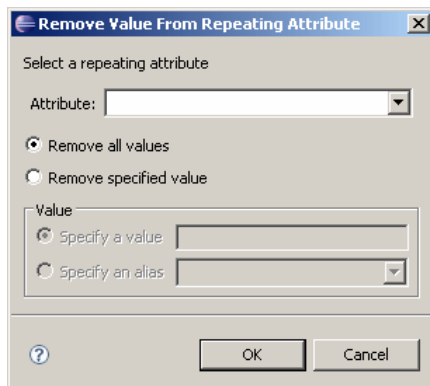
5. Click **OK** when you are finished.

## Removing repeating attributes values

Repeating attributes are attributes that can have more than one value. For example, the document object's authors attribute is a repeating attribute since a document can have more than one author.

### To remove one or more values from a repeating attribute:

1. In the lifecycle state diagram, click the state for which you want to remove one or more repeating attributes values.
2. Click the **Actions** tab in the **Properties** pane.  
The **Actions** page appears.
3. Click the **Remove value from repeating attribute(s)** link.  
The **Remove Value from Repeating Attribute** dialog appears.



4. Select the attribute and specify the value or alias you want to remove, as described in [Table 27, page 123](#).

**Table 27. Remove repeating attribute properties**

Property	Description
Attribute	The name of the repeating attribute. You can either type the name in the <b>Attribute</b> field or select a name from the drop-down list. The list only displays the attributes that are valid for the primary object you assigned to the lifecycle.
Remove all values	Select this option to remove all values from the repeating attribute.
Remove specified value	Select this option if you want to remove a specific value or alias to be removed, then enter the value or select the alias set in the Value section.
<b>Value</b>	The value of the repeating attribute.
Specify a value	Select this option if you want a specific value to be removed from the attribute, and enter the value.
Specify an alias	Select this option if you want a specific alias to be removed from the attribute, then select the alias from the drop-down list.

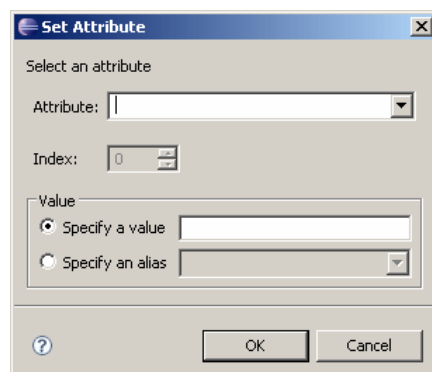
- Click **OK** when you are finished.

## Setting attributes

The **Set Attribute** action lets you specify attributes for a state. For example, the title or version of a document.

### To set an attribute for a lifecycle state:

- In the lifecycle state diagram, click the state for which you want to configure an attribute.
- Click the **Actions** tab in the **Properties** pane.  
The **Actions** page appears.
- Click the **Set attribute** link.  
The **Set Attribute** dialog appears.



4. Select the attribute, and enter an index and value for the attribute, as described in [Table 28, page 124](#).

**Table 28. Set attribute properties**

Property	Description
Attribute	The name of the attribute. You can either type the name in the <b>Attribute</b> field or select a name from the drop-down list. The list only displays the attributes that are valid for the primary object you assigned to the lifecycle.
Index	Stores the position of the attribute according to the index value that is entered. All attributes are stored in a list, therefore an index value must be stored with the attribute's name.
<b>Value</b>	The value of the attribute.
Specify a value	Select this option if you want the attribute to be stored as a value, and enter the value.
Specify an alias	Select this option if you want the attribute to be stored as an alias, then select the alias from the drop-down list. If the drop-down list does not show any aliases, no aliases have been configured for the project yet.  For more information about configuring aliases, see <a href="#">Chapter 6, Managing Alias Sets</a> .

5. Click **OK** when you are finished.

## Setting version labels

Version labels let you specify which version of a document can be attached to a lifecycle state.

### To specify a version label:

1. In the lifecycle state diagram, click the state for which you want to specify a document version.
2. Click the **Actions** tab in the **Properties** pane.  
The **Actions** page appears.
3. Click the **Set version label** link.  
The **Set Version Label** dialog appears.
4. Enter the version label in the **Version label** field, then click **OK** to save your changes.

## Removing version labels

Use the **Remove version label** link to remove version labels from a lifecycle state.

### To remove a version label:

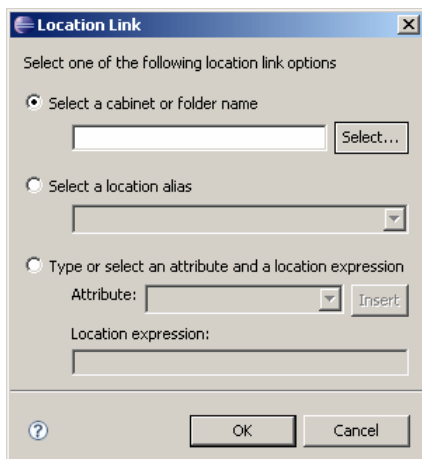
1. In the lifecycle state diagram, click the state for which you want to remove a document version.
2. Click the **Actions** tab in the **Properties** pane.  
The **Actions** page appears.
3. Click the **Remove version label** link.  
The **Remove Version Label** dialog appears.
4. Enter the version label in the **Version Label** field, then click **OK** to save your changes.

## Setting location links

Location links let you link a document to a specific location, such as a folder, a cabinet, an alias, or a local expression. The link is created when the document enters the state.

### To link a document to a specific location:

1. In the lifecycle state diagram, click the state for which you want to configure a location link.
2. Click the **Actions** tab in the **Properties** pane.  
The **Actions** page appears.
3. Click the **Set location link** link.  
The **Location Link** dialog appears.



4. Select one of the location link options and enter the location to which you want to link the document when it enters the state, as described in [Table 29, page 126](#).

**Table 29. Location link properties**

Property	Description
Select a cabinet or folder	Select this option to link the state to a cabinet or folder. Click <b>Select</b> . The <b>Folder Subtype Artifact</b> dialog appears. Select a cabinet or folder from the listbox.
Select a location alias	Select this option to link the state to a location alias. Enter the location alias or select an alias from the drop-down list.
Type or select an attribute and a location expression	Select this option to link the state to an attribute and a location expression.
Attribute	A string specifying the name of the attribute for the location expression. Enter the attribute name or select an attribute from the drop-down list. The list only displays the attributes that are valid for the primary object you assigned to the lifecycle.
Location expression	The dynamic location path associated with the attribute selected from the <b>Attribute</b> drop-down list. Enter the location expression. The location path is resolved at runtime.

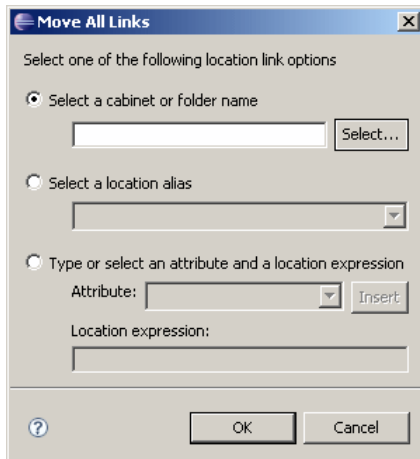
5. Click **OK** when you are finished.

## Moving all links

The **Set location to move all links** link lets you move all links to a specific location.

### To move all location links:

1. In the lifecycle state diagram, click the state for which you want to move all links.
2. Click the **Actions** tab in the **Properties** pane.  
The **Actions** page appears.
3. Click **Set location to move all links**.  
The **Move All Links** dialog appears.



4. Select one of the location link options and enter the location to which you want to move all links when the document enters the state, as described in [Table 30, page 127](#).

**Table 30. Move all links properties**

Property	Description
Select a cabinet or folder	Select this option to move all links to a cabinet or folder. Click <b>Select</b> . The <b>Folder Subtype Artifact</b> dialog appears. Select a cabinet or folder from the listbox.
Select a location alias	Select this option to move all links to a location alias. Enter the location alias or select an alias from the drop-down list.
Type or select an attribute and a location expression	Select this option to move all links to an attribute and a location expression.
Attribute	A string specifying the name of the attribute for the location expression. Enter the attribute name or select an attribute from the drop-down list. The list only displays the attributes that are valid for the primary object you assigned to the lifecycle.
Location expression	The dynamic location path associated with the attribute selected from the <b>Attribute</b> drop-down list. Enter the location expression. The location path is resolved at runtime.

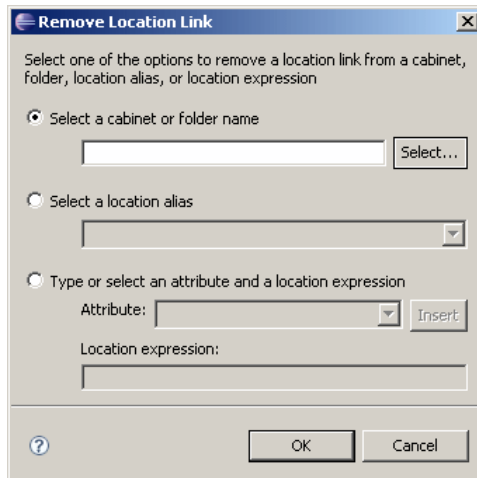
5. Click **OK** when you are finished.

## Removing location links

Use the **Remove Location Link** dialog to remove location links.

**To remove location links:**

1. In the lifecycle state diagram, click the state from which you want to remove a location link.
2. Click the **Actions** tab in the **Properties** pane.  
The **Actions** page appears.
3. Click the **Remove location link** link.  
The **Remove Location Link** dialog appears.



4. Select one of the location link options and enter the location from which you want to remove the document link when the document enters the state, as described in [Table 31, page 128](#).

**Table 31. Remove location link properties**

Property	Description
Select a cabinet or folder	Select this option to remove the link to a cabinet or folder. Click <b>Select</b> . The <b>Folder Subtype Artifact</b> dialog appears. Select a cabinet or folder from the listbox.
Select a location alias	Select this option to remove the link to a location alias. Enter the location alias or select an alias from the drop-down list.
Type or select an attribute and a location expression	Select this option to remove the link to an attribute and a location expression.
Attribute	A string specifying the name of the attribute for the location expression. Enter the attribute name or select an attribute from the drop-down list. The list only displays the attributes that are valid for the primary object you assigned to the lifecycle.
Location expression	The dynamic location path associated with the attribute selected from the <b>Attribute</b> drop-down list. Enter the location expression. The location path is resolved at runtime.



5. Click **OK** when you are finished.

## Assigning a document renderer

Composer uses Auto Render Pro for Windows or Macintosh to display a document attached to a lifecycle.

### To assign the rendering application for an attached document:

1. In the lifecycle state diagram, click the state for which you want to configure the rendering application.
2. Click the **Actions** tab in the **Properties** pane.  
The **Actions** page appears.
3. Click **Set rendition type**.  
The **Rendition Type** dialog appears.
4. Select one of the rendering options, as follows:
  - **Auto Render Pro for Windows**, if you are running Composer on a machine with a Windows operating system.
  - **Auto Render Pro for Macintosh**, if you are running Composer on a machine with a Macintosh operating system.
5. Click **OK** when you are finished.

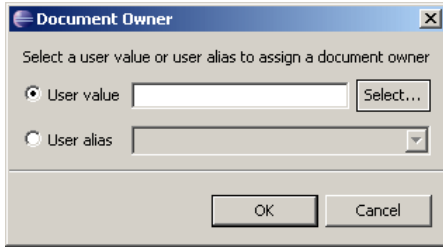
## Assigning document owners

By default, the owner of a document is the user who creates it. However, you can assign ownership to another user or a group. To assign another user as the owner, you must be a superuser. To assign a group as the owner of an object, you must either be a superuser or own the document and be a member of the group to which you are assigning ownership.

**Note:** You must have an alias set assigned to the lifecycle to assign an owner to a document. You can assign an alias set in the **General** tab of the lifecycle properties page ([Figure 5, page 114](#)).

### To assign an owner to a document:

1. In the lifecycle state diagram, click the state for which you want to assign a document owner. The **Properties** pane appears below the lifecycle editor.
2. Click the **Actions** tab in the **Properties** pane. The **Actions** page appears.
3. Click the **Set owner** link. The **Document Owner** dialog appears.



4. Select one of the user options, as described in [Table 32, page 130](#).

**Table 32. Document owner properties**

Property	Description
User value	Select this option assign a user value. Click <b>Select</b> . The <b>Principal (User or Group) Installation Parameter</b> dialog appears. Select a user value from the listbox or click <b>New</b> to create a user value.
User alias	Use this option to assign a user alias. Select an alias from the drop-down list.

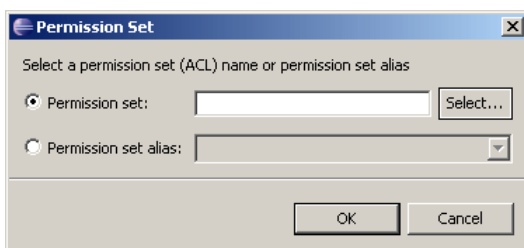
5. Click **OK** to save your changes.

## Setting permission sets

Permission sets (also known as ACLs, or access control lists) specify the operations (such as read, edit, create a version, or delete) users can perform on a document attached to a lifecycle.

### To assign a permission set to a lifecycle state:

1. In the lifecycle state diagram, click the state for which you want to assign a permission set.
2. Click the **Actions** tab in the **Properties** pane.  
The **Actions** page appears.
3. Click the **Set permission set** link.  
The **Permission Set** dialog appears.



4. Select one of the permission set options, as described in [Table 33, page 131](#).

**Table 33. Permission set properties**

Property	Description
Permission set	Select this option to assign a permission set to the lifecycle state. Click <b>Select</b> to open the <b>Permission Set (ACL) Template</b> artifact dialog. Select a permission set from the list or click <b>New</b> to create a permission set. For information about permission sets, see <a href="#">Chapter 13, Managing Permission Sets (ACLs)</a> .
Permission set alias	Select this option to assign a permission set alias and select a permission set alias from the drop-down list. For information about permission sets, see <a href="#">Chapter 13, Managing Permission Sets (ACLs)</a> .

## Configuring post-change information

A post-change procedure executes after the state transition is complete. When the part of a transition that occurs within the transaction is complete, the system executes the post-change procedure. Failure of any part of the post-change procedure does not prevent the transition from succeeding.

### To assign a post-change procedure to a lifecycle state:

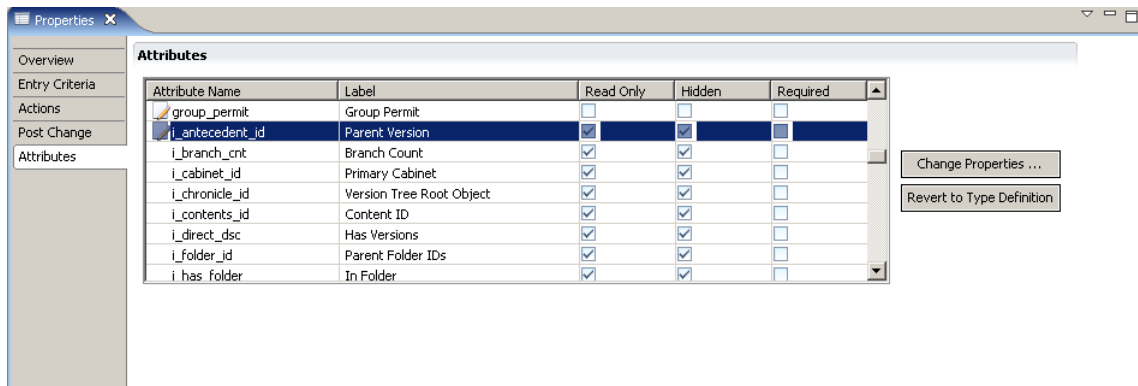
1. In the lifecycle state diagram, click the state to which you want to assign a post-change procedure.
2. Click the **Post Change** tab in the **Properties** pane.  
The **Procedure** page appears.
3. Click **Select**.  
The **Procedure Artifact** dialog appears.
4. Select a post-change procedure from the list or click **New** to create a post-change procedure.  
For more information about creating procedures, see [Chapter 14, Managing Procedures](#).

## Configuring state attributes

State attributes include labels, help text, comment, and attribute properties for this state. The required and cannot be blank properties are checked when the client application validates an object (which typically occurs on saving or checking in an object), not when it enters the state.

### To configure state attributes:

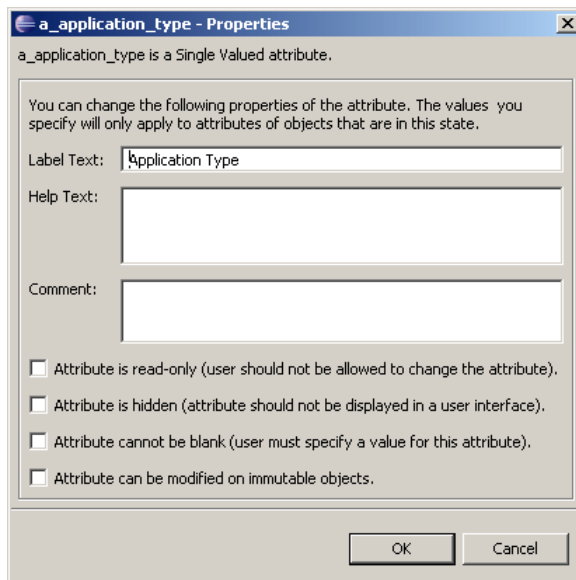
1. In the lifecycle state diagram, click the state for which you want to configure attributes.
2. Click the **Attributes** tab in the **Properties** pane.  
The **Attributes** page appears.



The page only displays the attributes that are valid for the primary object you assign to the lifecycle.

3. Select an attribute from the **Attributes** table and click **Change Properties...** to change any of the attribute's properties.

The **Properties** dialog for that attribute appears.



4. Configure the attribute's properties by selecting any of the available options, then click **OK**.  
You can revert back to the original type definition for the attribute by clicking **Revert to Type Definition**.

## Deleting a state

### To delete a lifecycle state:

1. In the lifecycle state diagram, select the state you want to delete.

2. Delete the state using one of the following methods:
  - Right-click the state and select **Delete** from the drop-down menu.
  - Select **File > Delete** from the Composer menu.

## Deleting a lifecycle

### To delete a lifecycle:

1. Locate the lifecycle in the **Documentum Navigator** view.
2. Right-click the lifecycle and select **Delete** from the pop-up menu.



## Managing Methods and Jobs

This chapter contains the following topics:

- [Methods and jobs, page 135](#)
- [Creating a method , page 135](#)
- [Creating a job, page 137](#)

### Methods and jobs

Methods are executable programs that are represented by method objects in the repository. The program can be a Docbasic script, a Java method, or a program written in another programming language such as C++.

Jobs automate the execution of a method, for example how to transfer content from one storage place to another. The attributes of a job define the execution schedule and turn execution on or off.

### Creating a method

Use the **Method** editor to create a method.

#### To create a method:

1. Open the **Select a wizard** dialog in one of the following ways:
  - From the Composer menu, select **File > New > Other**.
  - In your Composer project, expand the **Artifacts** folder and right-click **Method**. Select **New > Other**.

The **Select a wizard** dialog appears.

2. Double-click the Documentum folder to expand it, then select **Method**.  
The **New Documentum Artifact – Name and Location** dialog appears.
3. Enter a name for the new method or accept the default name, then click **Finish**.  
The **Method** editor appears.

**General**  
Define an executable method.

Name:

Type: ☒ Java  
☐ DocBasic  
☐ dmawk  
☐ Program

Command:

**Run Controls**  
Set the runtime options of the method.

☐ Run Asynchronously  
☒ Run Synchronously

Timeout (seconds):

☐ Run as the server  
☐ Trace launch  
☐ Use method server  
☐ Launch directly  
☐ Use as workflow method

Overview

4. Enter the properties for the method, as described in [Table 34, page 136](#).

**Table 34. Method artifact properties**

Parameter	Description
<b>General</b>	
Name	A string specifying the name of the method. Do not use the format <code>dm_methodname</code> to name the method. This naming convention is reserved for default Documentum objects.
Type	Specifies the language or program used for this method. Select one of the following types: <ul style="list-style-type: none"> <li>• <b>Java</b> — The method is written in Java and the Java method server is executing the method.</li> <li>• <b>Docbasic</b> — The method is written in Docbasic.</li> <li>• <b>dmawk</b> — The method is written in dmawk.</li> <li>• <b>Program</b> — The method is a program.</li> </ul>
Command	The command that launches the method.
<b>Run Controls</b>	
Run Asynchronously	Specifies whether the procedure is run asynchronously or not. This parameter is ignored if the method is launched on the Method Server or Content Server and <code>SAVE_RESULTS</code> is set to <code>TRUE</code> on the command line.
Run Synchronously	Specifies whether the method is run synchronously.
Timeout	Specifies the minimum, default, and maximum amount of time before the methods times out.
Run as the server	Specifies whether the method is run as the server account. If selected, the method is run as the server account.



Parameter	Description
Trace launch	Specifies whether internal trace messages generated by the executing program are logged. If selected, the messages are stored in the session log.
Use method server	Specifies whether to use the Method Server or Application Server to execute Dmbasic or Java methods. If selected, the Method Server or application server is used. If not selected, the Content Server is used.
Launch directly	Specifies whether the program is executed by the operating system or exec API call. If selected, the server uses the exec call to execute the procedure. If the checkbox is cleared, the server uses the system call to execute the procedure.
Use as workflow method	Specifies whether this method is used in a workflow.

5. Save your changes.

## Creating a job

A job automates the execution of a method.

### To create a job:

1. Open the **Select a wizard** dialog in one of the following ways:
  - From the Composer menu, select **File > New > Other**.
  - In your Composer project, expand the **Artifacts** folder and right-click **Job**. Select **New > Other**.

The **Select a wizard** dialog appears.

2. Double-click the Documentum folder to expand it, then select **Job**. The New Documentum Artifact – Name and Location dialog appears.
3. Enter a name for the new job or accept the default name, then click **Finish**.  
The **Job** editor appears.

4. Enter the properties for the job in the Info, Job Schedule, and Run Interval sections, as described in [Table 35, page 138](#).

**Table 35. Job properties**

Parameter	Description
<b>Info</b>	
Name	A string specifying the name of the job.
Subject	A comment or description of the job.
Method	The method that this job is automating. Click <b>Select</b> and select a method from the <b>Documentum Method Artifact</b> dialog or click <b>Method</b> to create a method for this job.
Method data	Data that is used by method associated with job. This property is available for the method to write/read as needed during its execution. Enter the data in the method data field and click <b>Add</b> .
Standard arguments	<p>Select to pass the standard arguments to the method. The standard arguments are:</p> <ul style="list-style-type: none"> <li>• <b>Repository owner</b></li> <li>• <b>Repository name</b></li> <li>• <b>Job ID</b></li> <li>• <b>Trace level</b></li> </ul>

Parameter	Description
Custom arguments	Select to pass one or more custom arguments to the method. Enter the argument in the custom arguments field and click <b>Add</b> .
Deactivate on failure	Select to direct the application to stop running the job if the underlying method fails.
Trace level	Controls tracing for the method. Any value other than 0 turns on tracing. By default the trace level is set to 0.
Make this job active	Select to activate the job.
<b>Job Schedule</b>	
Max runs	Specifies the maximum number of times the job is run.
Run once upon saving, then run as scheduled	Select to run the job immediately after you save it, then return to the configured schedule.
<b>Run Interval</b>	
Execute every	Specifies the number of times this job is run every minute, hour, day, or week.
Execute on	Specifies a day on which this job is run. The job can be run on the same day once every week, once every month, or once every year.

5. Save your changes.



## Managing Modules

This chapter contains the following topics:

- [Modules, page 141](#)
- [Creating a module, page 141](#)
- [Configuring module deployment , page 144](#)
- [Configuring the module runtime environment, page 145](#)

## Modules

A module consists of executable business logic and supporting material, such as third-party software and documentation. A module is comprised of the JAR files that contain the implementation classes and the interface classes for the behavior the module implements, and any interface classes on which the module depends. The module might also include Java libraries and documentation.

There are three types of modules, as follows:

- Service-based modules (SBOs)

An SBO provides functionality that is not specific to a particular object type or repository. For example, an SBO can be used to customize a user's inbox.

- Type-based modules (TBOs)

A TBO provides functionality that is specific to an object type. For example, a TBO can be used to validate the title, subject, and keywords properties of a custom document subtype.

- Aspect

An aspect provides functionality that is applicable to specific objects. For example, an aspect can be used to set the value of a one property based on the value of another property. An aspect module is created using a different editor, the **Aspect Module** editor, as described in [Creating an aspect type, page 87](#).

## Creating a module

Use the **Module** editor to create a module.

### To create a module:

1. Open the **Select a wizard** dialog in one of the following ways:
  - From the Composer menu, select **File > New > Other**.
  - In your Composer project, expand the **Artifacts** folder and right-click **Module**. Select **New > Other**.

The **Select a wizard** dialog appears.

2. Double-click the Documentum folder to expand it, then select **Module**.

The **New Documentum Artifact – Name and Location** dialog appears.

3. Enter a name for the new module or accept the default name, then click **Finish**.

The **Module** editor appears with the **General** tab selected.

4. Enter the required and optional properties in the **Info**, **Description**, **Required Modules**, **Deployment**, **Javadoc**, and **Core JARs** sections, as described in [Table 36, page 142](#).

**Table 36. Properties in General tab**

Property	Description
<b>Info</b>	
Name	<p>A string specifying the name of the module. Required parameter. Enter the module name associated with the module's type, as follows:</p> <ul style="list-style-type: none"> <li>• <b>SBO</b> module — Enter the fully qualified primary interface name of the SBO.</li> <li>• <b>TBO</b> module — Enter the name of the corresponding object type. The name can have up to 255 characters.</li> </ul>

Property	Description
Type	<p>A string specifying the type of the module. Required parameter. Enter the module type or select the type from the drop-down list. Composer provides the following standard module types:</p> <ul style="list-style-type: none"> <li>• <b>Standard Module</b> — Provides a generic module.</li> <li>• <b>TBO</b> — Provides functionality that is specific to an object type. For example, a TBO can be used to validate the title, subject, and keywords properties of a custom document subtype.</li> <li>• <b>SBO</b> — Provides functionality that is not specific to a particular object type or repository. For example, an SBO can be used to customize a user's inbox.</li> </ul>
<b>Description</b>	
Author	Contact information for the module author. Optional parameter.
Description	Description for the module, not exceeding 255 characters. Optional parameter.
<b>Required Modules</b>	
	<p>Specifies modules that this module requires to function properly.</p> <p>Click <b>Add</b> to open the <b>Module Artifact</b> dialog. Select a module from the listbox and click <b>OK</b>, or click <b>New</b> to create a module.</p>
<b>Javadoc</b>	
	<p>Specifies Javadocs and other resources that can be downloaded with the module at runtime.</p> <p>Click <b>Select</b> to open the <b>SysObject Subtype Artifact</b> dialog. Select a SysObject that contains the Javadoc or resource content from the list or click <b>New</b> to create a SysObject that contains the content to be downloaded.</p> <p>The Java doc must be a zip file with content.</p>
<b>Core JARs</b>	
Implementation JARs	<p>Implementation of the module. Required parameter.</p> <p>Click <b>Add</b> to add implementation JARs from your local machine.</p>

Property	Description
Class name	Primary Java implementation class for the module. Required parameter.
Interface JARs	TBOs must implement the IDfBusinessObject interface, SBOs must implement the IDfService interface, and all modules must implement the IDfModule interface.  Java interfaces that this module implements. Optional parameter.  Click <b>Add</b> to add interface JARs from your local machine.

- Click the **Deployment** tab and configure the module deployment options as described in [Configuring module deployment , page 144](#).
- Click the **Runtime** tab to configure the runtime environment for this module, as described in [Configuring the module runtime environment, page 145](#).

## Configuring module deployment

The **Deployment** tab lets you link Java libraries and other modules to the module you are creating or editing.

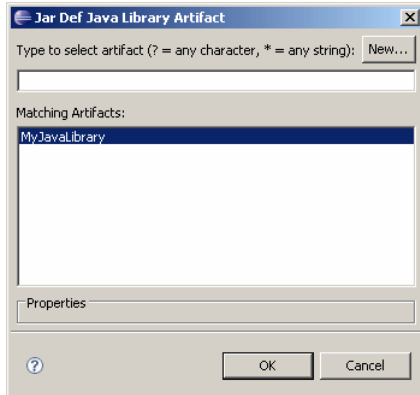
### To configure module deployment:

- Click the **Deployment** tab in the **Module** editor.  
The **Deployment** view appears.

The screenshot shows the 'Deployment' tab in the Module editor. It features a 'Java Libraries' section with a text area and 'Add...', 'Remove', and 'Edit...' buttons. Below it is an 'Attachments' section with a table and 'Add...' and 'Remove' buttons. To the right is a 'Logging' section with a text area for a post-download message and a 'Log level' dropdown menu set to 'WARN'. At the bottom, there are tabs for 'General', 'Deployment', and 'Runtime'.

- In the **Java Libraries** section, click **Add** to add Java libraries for this module.  
The **Jar Def Java Library Artifact** dialog appears.





Select a Java library from the listbox and click **OK** or click **New** to create a Java Library. You can only link existing Java libraries into this module. You cannot modify an existing library that is shared by multiple modules. For more information about creating a Java Library, see [Linking and configuring a Java Library, page 109](#).

3. In the **Attachments** section, specify Javadocs and other resources that should be available for download when the module is deployed.
4. In the **Logging** section, specify a post-download message and select a log level for the message. The log level can have the following values:
  - **WARN** — The post-download message is logged as a warning.
  - **NONE** — The post-download message is not logged.
  - **INFO** — The post-download message is logged as an informational message.
  - **DEBUG** — The post-download message is logged at debug level.
5. Save your changes.

## Configuring the module runtime environment

The runtime environment lets you configure optional properties that are executed at runtime, such as version requirements, Java system properties, statically deployed classes, and local resources.

### To configure the runtime environment:

1. Click the **Runtime** tab in the **Module** editor.  
The **Runtime** view appears.

**Runtime Environment**

**Version Requirements**  
Specify the minimum DFC and Java VM versions required at runtime  
Min DFC version:   
Min VM version:

**Java System Properties**  
Specify Java system properties to be defined at runtime

Name	Value
new name	new value

Add... Remove

**Statically Deployed Classes**  
Specify classes that must be available at runtime. These classes are dynamically loaded by the application. They are not distributed with the application.  
Fully-qualified class name:  Add > Remove

**Local Resources**  
Specify additional resources that must be present in the file system at runtime  
File path relative to deployment:  Add > Remove

Overview Dependencies Runtime

- Specify the version requirements, Java system properties, statically deployed classes, and local resources, as described in [Table 37, page 146](#).

**Table 37. Module runtime environment properties**

Property	Description
<b>Version Requirements</b>	This section lets you specify the DFC and Java VM versions required on the client for the module to function properly.
Min DFC version	The minimum DFC version on the client machine for this module to work properly.
Min VM version	The minimum Java VM version on the client machine for this module to work properly.
<b>Java System Properties</b>	This section lets you specify Java system properties as name-value pairs. When the module is downloaded, the client machine is checked to see if all the specified Java properties match the properties on the client machine. Click <b>Add</b> to enter placeholders for the name and value of the Java system property, then click the <b>Name</b> and the <b>Value</b> fields to modify the property name and value.
Name	Name of the Java system property.
Value	Corresponding value for the Java system property name.
<b>Statically Deployed Classes</b>	This section lets you specify static Java classes that are required for the module to function properly. When the module is downloaded, the class path is checked for the specified Java classes.
Fully qualified class name	Fully qualified Java class names. Enter the class name and click <b>Add</b> .

Property	Description
<b>Local Resources</b>	This section lets you specify files that are required on the local machine for the module to function properly. When the module is downloaded, the client machine is checked for the specified files specified.
File path relative to deployment	Full file path. Enter the file name and path and click <b>Add</b> .

3. Save your changes.



## Managing Permission Sets (ACLs)

This chapter contains the following topics:

- [Permissions, permission sets, and permission set templates, page 149](#)
- [Creating a permission set template, page 151](#)
- [Creating a regular or a public permission set, page 153](#)

### Permissions, permission sets, and permission set templates

Access to folders and documents in a repository is subject to an organization's security restrictions. All content in the repository is associated with object permissions, which determine the access users have to each object in the repository such as a file, folder, or cabinet and governs their ability to perform specific actions. There are two categories of object permissions:

- **Basic**—Required for each object in the repository.
- **Extended**—Optional.

Permission sets (also known as access control lists, or ACLs) are configurations of basic and extended permissions assigned to objects in the repository that lists users and user groups and the actions they can perform. Each repository object has a permission set that defines the object-level permissions applied to it, including who can access the object. Depending on the permissions, users can create new objects, perform file-management actions such as importing, copying, or linking files, and start processes, such as sending files to workflows.

ACLs are the mechanism that Content Server uses to impose object-level permissions on SysObjects. A permission set has one or more entries that identify a user or group and the object-level permissions assigned to user or group. Composer lets you create permission set templates, regular, and public ACLs, as follows:

- **Template**—Creates a permission set template. Template ACLs are used to make applications, workflows, and lifecycles portable. For example, an application that uses a template ACL could be used by various departments within an enterprise because the users or groups within the ACL entries are not defined until the ACL is assigned to an actual document.
- **Public**—Creates a public ACL that can be used by anyone in a repository. Public ACLs are available for use by any user in the repository.
- **Regular**—Creates a regular ACL that can only be used by the user or group that creates it.

## Basic permissions

Basic permissions grant the ability to access and manipulate an object's content. The seven basic permission levels are hierarchical and each higher access level includes the capabilities of the preceding access levels. For example, a user with Relate permission also has Read and Browse. The basic permission are described in [Table 38, page 150](#).

**Table 38. Basic permissions**

Basic Permission	Description
None	No access to the object is permitted.
Browse	Users can view the object's properties but not the object's content.
Read	Users can view both the properties and content of the object.
Relate	Users can do the above and add annotations to the object.
Version	Users can do the above and modify the object's content and check in a new version of the item (with a new version number). Users cannot overwrite an existing version or edit the item's properties.
Write	Users can do the above and edit object properties and check in the object as the same version.
Delete	Users can do all the above and delete objects.

## Extended permissions

Extended permissions are optional, grant the ability to perform specific actions against an object, and are assigned in addition to basic permissions. The six levels of extended permissions are not hierarchical, so each must be assigned explicitly. The extended permissions are described in [Table 39, page 150](#).

**Table 39. Extended permissions**

Extended Permission	Description
Execute Procedure	Superusers can change the owner of an item and use Run Procedure to run external procedures on certain object types. A procedure is a Docbasic program stored in the repository as a dm_procedure object.
Change Location	Users can move an object from one folder to another in the repository. A user also must have Write permission to move the object. To link an object, a user also must have Browse permission.
Change State	Users can change the state of an item with a lifecycle applied to it.
Change Permissions	Users can modify the basic permissions of an object.

Extended Permission	Description
Change Ownership	Users can change the owner of the object. If the user is not the object owner or a Superuser, they also must have Write permission.
Extended Delete	Users can only delete the object. For example, you might want a user to delete documents but not read them. This is useful for Records Management applications where discrete permissions are common.

## Creating a permission set template

Template ACLs are used to make applications, workflows, and lifecycles portable. For example, an application that uses a template ACL could be used by various departments within an enterprise because the users or groups within the ACL entries are not defined until the ACL is assigned to an actual document.

### To create a permission set template:

1. Open the **Select a wizard** dialog in one of the following ways:
  - From the Composer menu, select **File > New > Other**.
  - In your Composer project, expand the **Artifacts** folder and right-click **Permission Sets**. Select **New > Other**.

The **Select a wizard** dialog appears.

2. Select **Documentum > Permission Set**, then click **Next**.

The **New ACL Artifact** wizard appears.

3. Enter the folder path and name of the project for which you want to create a permission set in the **Folder:** field, or click **Browse** to select the project from a folder list.

4. Enter a name for the permission set in the **Artifact name:** field.
5. Verify that **Template** is selected in the **ACL Class** field, then click **Finish**.  
The **Permission Set Template** editor appears.

The new permission set contains two default alias entries in the **All Users and Groups** section, as follows:

- **dm\_owner**—The owner of the permission set template.
- **dm\_world**—All repository users.

You cannot delete these default entries from a permission set template.

6. Enter a name and an optional description for the permission set template in the **General** section.
7. Select the **dm\_user** or **dm\_world** alias in the **All Users and Groups** section or click **Add Alias** to add a new alias.

The **ACL Entry Details** section appears.

Extended Permissions	Description
<input type="checkbox"/> Execute Procedure	Superusers can change the owner of an item and can use Execute Procedure to run external procedures on certain item types
<input type="checkbox"/> Change Location	Users with Change Location permission can move an item in the repository
<input type="checkbox"/> Change State	Users with Change State permission can change the state of an item that has a lifecycle applied to it
<input type="checkbox"/> Change Permissions	Users with Change Permissions can modify the basic permissions of an item
<input type="checkbox"/> Change Ownership	Users with Change Ownership permission can change the owner of an item
<input type="checkbox"/> Extended Delete	Users with the Delete Object extended permission have the right to only delete the object

8. Specify the name and permissions for the alias you selected, as described in [Table 40, page 153](#).



**Table 40. ACL entry details – Permission Set Template**

Parameter	Description
Owner Alias	A string specifying the owner for the alias. Click <b>Select</b> to select an owner for the ACL entry. The <b>Documentum AliasSet Artifact</b> dialog appears. Select an alias owner from the list. If the list is empty, create an alias first. For more information about creating an alias, see <a href="#">Creating an alias set, page 81</a> .
Permissions	Specifies the permissions for the alias. Select a permission from the drop-down list and optionally assign extended permission by checking the associated option in the <b>Extended Permissions</b> column. For more information about permissions, see <a href="#">Basic permissions, page 150</a> and <a href="#">Extended permissions, page 150</a> .

9. Save your changes.

## Creating a regular or a public permission set

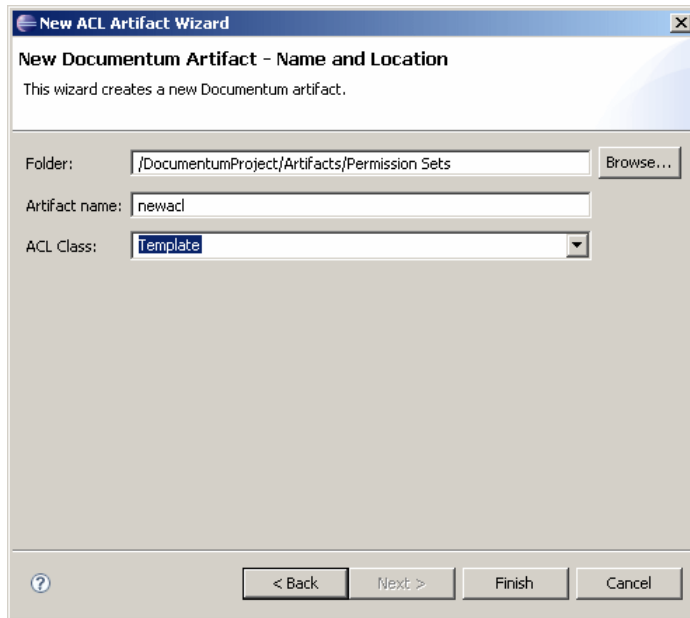
Regular ACLs can only be used by the user or group that creates it, while public ACLs can be used by any user or group in the repository.

### To create a regular or a public permission set:

1. Open the **Select a wizard** dialog in one of the following ways:
  - From the Composer menu, select **File > New > Other**.
  - In your Documentum project, expand the **Artifacts** folder and right-click **Permission Sets**. Select **New > Other**.

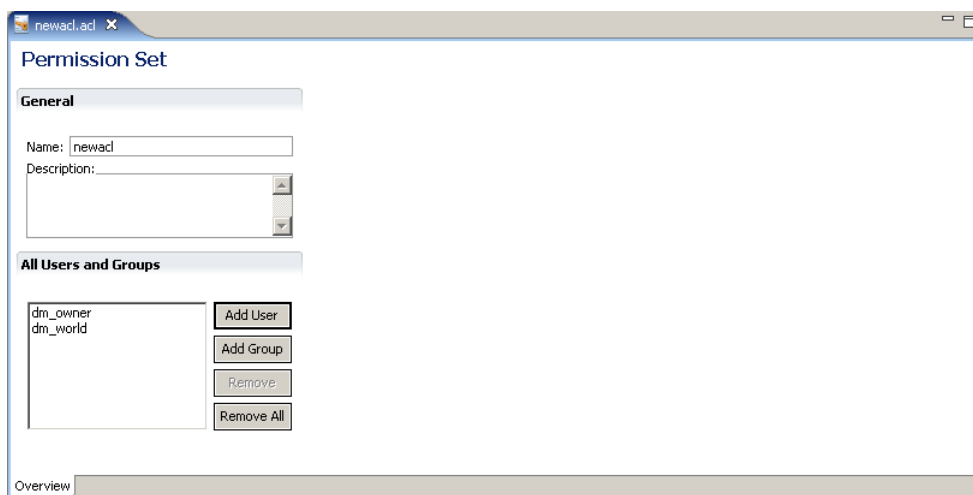
The **Select a wizard** dialog appears.

2. Select **Documentum > Permission Set**, then click **Next**.  
The **New ACL Artifact Wizard** appears.



3. Enter the folder path and name of the project for which you want to create a permission set in the **Folder:** field, or click **Browse** to select the project from a folder list.
4. Enter a name for the permission set in the **Artifact name:** field.
5. Select **Regular** or **Public** from the **ACL class** drop-down list, depending on what type of ACL you want to create, then click **Finish**.

The **Permission Set** editor appears.



The new permission set contains two default ACL entries in the **All Users and Groups** section, as follows:

- **dm\_owner**—The owner of the permission set.
- **dm\_world**—All repository users.

You cannot delete these default entries from a permission set.

6. Enter a name and an optional description for the permission set in the **General** section.

7. Select the dm\_user or dm\_world ACL entry in the **All Users and Groups** section or click
  - **Add User** to add a new user ACL
  - **Add Group** to add a new group ACL

The **ACL Entry Details** section appears.

The screenshot shows a window titled 'newacl.ac' with a 'Permission Set' dialog. The 'General' tab is selected, showing 'Name: newacl' and a 'Description' field. Below this is the 'All Users and Groups' section with a listbox containing 'New ACL User', 'New ACL Group', 'dm\_owner', and 'dm\_world', and buttons for 'Add User', 'Add Group', 'Remove', and 'Remove All'. To the right is the 'ACL Entry Details' section, which includes 'Owner User:' with a 'Select...' button, 'Permissions:' set to '<unspecified>', and a table of 'Extended Permissions'.

Extended Permissions	Description
<input type="checkbox"/> Execute Procedure	Superusers can change the owner of an item and can use Execute Procedure to run external procedures on certain item types
<input type="checkbox"/> Change Location	Users with Change Location permission can move an item in the repository
<input type="checkbox"/> Change State	Users with Change State permission can change the state of an item that has a lifecycle applied to it
<input type="checkbox"/> Change Permissions	Users with Change Permissions can modify the basic permissions of an item
<input type="checkbox"/> Change Ownership	Users with Change Ownership permission can change the owner of an item
<input type="checkbox"/> Extended Delete	Users with the Delete Object extended permission have the right to only delete the object

8. Specify the name and permissions for the ACL entry you selected, as described in [Table 41](#), [page 155](#).

**Table 41. ACL entry details – Permission Set**

Parameter	Description
Owner UserOwner Group	A string specifying the owner for the ACL entry. Click <b>Select</b> to select an owner for the ACL entry. The <b>User Installation Parameter</b> or <b>Group Installation Parameter</b> dialog appears, depending on whether you are adding a user ACL or a group ACL. Select an owner from the list. If the listbox in the <b>Installation Parameter</b> dialog is empty or does not contain the desired user or group, create an owner in the form of a user or group installation parameter. You cannot add users or groups directly to the ACL. Create an installation parameter for each group or user that you want to add to the ACL and specify the value of the group or user in that installation parameter. You can then specify the installation parameter in the ACL. For more information about creating an ACL entry owner, see <a href="#">Creating an ACL entry owner</a> , <a href="#">page 156</a> .
Permissions	Specifies the permissions for the ACL entry. Select a permission from the drop-down list and optionally assign extended permission by checking the associated option in the <b>Extended Permissions</b> column. For more information about permissions, see <a href="#">Basic permissions</a> , <a href="#">page 150</a> and <a href="#">Extended permissions</a> , <a href="#">page 150</a> .

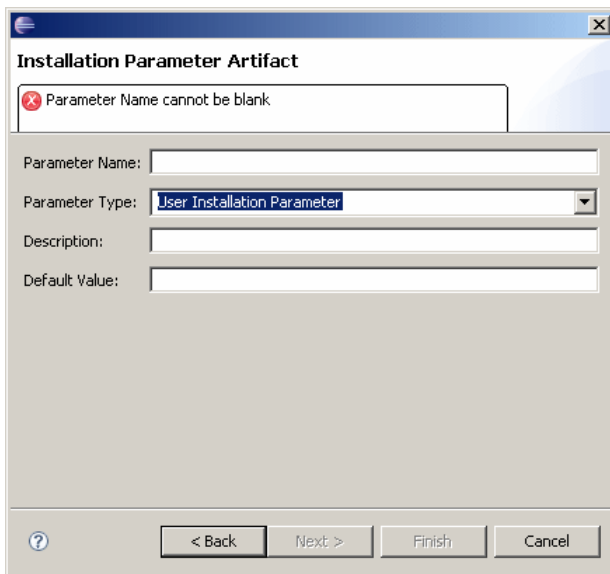
9. Save your changes.

## Creating an ACL entry owner

Every ACL entry requires an owner, which can be either a user or a group of users. Composer lets you create an owner parameter in form of a user or a group installation parameter. The parameter can be mapped to the associated owner in a repository when the project is installed into the repository.

### To create a user or group owner for an ACL entry:

1. Click **Select** in the **ACL Entry Details** section of the **Permission Set** editor.  
The **User or Group Installation Parameter** dialog appears.
2. Click **New**.  
The **New Documentum Artifact** dialog appears.
3. Accept the default folder location and artifact name, and click **Next**.  
The **Installation Artifact** dialog appears.

The image shows a Windows-style dialog box titled "Installation Parameter Artifact". At the top, there is a red error icon and the text "Parameter Name cannot be blank". Below this, there are four input fields: "Parameter Name:" (which is empty), "Parameter Type:" (a dropdown menu showing "User Installation Parameter"), "Description:" (empty), and "Default Value:" (empty). At the bottom of the dialog, there is a row of four buttons: a help button (question mark icon), "< Back", "Next >", and "Cancel".

4. Enter a name for the ACL entry owner in the **Parameter name** field. You may also enter an optional description and a default value.
5. Click **Next**. The new owner name appears in the **Matching artifacts** list.
6. Click **OK** to save your changes.

# Managing Procedures

This chapter contains the following topics:

- [Procedures, page 157](#)
- [Creating a procedure, page 157](#)

## Procedures

Procedures are applications that extend or customize the behavior of Documentum clients or Content Server. Depending on where they are stored in the repository, procedures can be executed automatically when a user connects to a repository, or on demand when users select a menu item. Procedures are written in a proprietary Documentum language called Docbasic, which is based on Visual Basic.

For more information about Docbasic, see the *Docbasic Reference Manual*.

## Creating a procedure

Use the **Procedure** editor to create a procedure.

### To create a procedure:

1. Open the **Select a wizard** dialog in one of the following ways:
  - From the Composer menu, select **File > New > Other**.
  - In your Composer project, expand the **Artifacts** folder and right-click **Procedures**. Select **New > Other**.

The **Select a wizard** dialog appears.

2. Select **Documentum > Procedure**, then click **Next**.

The **New Documentum Resource - Name and Location** dialog appears.

3. Enter the folder path and name of the project for which you want to create a procedure in the **Folder:** field, or click **Browse** to select the project from a folder list.
4. Enter a file name for the procedure in the **Artifact name:** field, then click **Finish**.

The **Procedure** editor appears.

### General

The screenshot shows a web-based configuration interface for a procedure. It has two main sections: 'Info' and 'DmBasic Content'. The 'Info' section contains a 'Name' field with the text 'newprocedure' and a 'User runnable' checkbox which is currently unchecked. The 'DmBasic Content' section has a text area for entering code and a 'Load...' button to the right. The text area is empty and has a scrollbar on the right side.

**Info**

Name:

☐ User runnable

**DmBasic Content**

Enter your DmBasic content or load existing DmBasic content into the text area below.

5. Enter a name for the procedure in the **Name** field or accept the default name.
6. Check the **User runnable** checkbox if a user is allowed to execute the procedure in the associated client application.
7. Enter the Docbasic code for the procedure in the **Docbasic Content** section or click **Load** to load the procedure code from a local file.

# Managing Relation Types

This chapter contains the following topics:

- [Relation types, page 159](#)
- [Creating a relation type, page 159](#)

## Relation types

A relation type defines the relationship between two objects in a repository. In general, when two objects are connected by a relationship, one is considered the parent objects and the other is considered the child.

A relation type describes how one item is related to another. There are two relation types, as follows:

- **Ad hoc**

This relation type can be added, modified and deleted by users.

- **System**

This relation type cannot be manipulated by users. For example, a relationship between a file and its thumbnail is a system relation type.

## Creating a relation type

Use the **Relation Type** editor to create a relation type or modify an existing relation type.

### To create a relation type:

1. Open the **Select a wizard** dialog in one of the following ways:
  - From the Composer menu, select **File > New > Other**.
  - In your Composer project, expand the **Artifacts** folder and right-click **Relation Types**. Select **New > Other**.

The **Select a wizard** dialog appears.

2. Select **Documentum > Relation Type**, then click **Next**.

The **New Documentum Resource - Name and Location** dialog appears.

- Enter the folder path and name of the project for which you want to create a relation type in the **Folder:** field, or click **Browse** to select the project from a folder list.
- Enter a file name for the relation type in the **Artifact name:** field, then click **Next**.
- Enter a name for the relation type in the **Relation type name:** field, then click **Finish**.  
The **Relation Type** editor appears.

#### Relation Type

The screenshot shows the 'Relation Type' editor with two main sections: 'General' and 'Parent and Child'.

**General Section:**

- Name:** A text input field.
- Description:** A large text area with a vertical scrollbar.
- Security type:** A dropdown menu with 'None' selected.
- Referential integrity:** A dropdown menu with 'Allow delete' selected.

**Parent and Child Section:**

- Child Type:** A text input field with a 'Select...' button.
- Parent type:** A text input field with a 'Select...' button.
- Relationship direction:** A dropdown menu with 'From Parent to Child' selected.
- Permanent link:** A checkbox that is checked. Below it are two radio buttons:
  - ☐ The child object is copied if the parent is copied.
  - ☒ The child object is not copied.
- Child-to-Parent label:** A section with a text input field, an 'Add >' button, and a 'Remove' button.
- Parent-to-Child label:** A section with a text input field, an 'Add >' button, and a 'Remove' button.

- Enter the relation type properties in the **General** and **Parent and Child** sections, as described in [Table 42, page 160](#).

**Table 42. Relation type properties**

Property	Description
<b>General</b>	
Name	A string specifying the name of the relation type. The name can be up to 255 characters long.
Description	A string describing the relation type. The description can be up to 250 characters long.
Security type	<p>A string specifying the security level for the relation type. The security type can have the following values:</p> <ul style="list-style-type: none"> <li>• <b>System</b> — Only users with superuser or system administrator privileges can create, modify, or drop this relation type.</li> <li>• <b>Parent</b> — This relation type inherits the security level from its parent.</li> <li>• <b>Child</b> — This relation type inherits the security level from its child.</li> <li>• <b>None</b> — This relation type has no security level.</li> </ul>



Property	Description
Referential integrity	<p>Specifies how the referential integrity is enforced. Select one of the following values from the drop-down list:</p> <ul style="list-style-type: none"> <li>• <b>Allow delete</b></li> <li>• <b>Restrict delete</b></li> <li>• <b>Cascade delete</b></li> </ul> <p>The default referential integrity value is <b>Allow delete</b>.</p>
<b>Parent and Child</b>	
Child type	<p>A string with a maximum of 32 characters specifying the object type for a child object. The child type is an optional relation type property. You can specify a child type in one of the following ways:</p> <ul style="list-style-type: none"> <li>• Type the name of the child type in the <b>Child type:</b> field.</li> <li>• Click <b>Select</b> to select a child type. The <b>Select Type Artifact</b> dialog appears. Select a child type from the listbox.</li> <li>• Click the <b>Child type:</b> link to create a child type. The <b>Documentum Artifact - Name and Location</b> wizard appears. For more information about creating an artifact, see <a href="#">Creating an artifact</a>, page 30.</li> </ul>
Parent type	<p>A string with a maximum of 32 characters specifying the object type of a valid parent object. The parent type is an optional relation type property. You can specify a child type in one of the following ways:</p> <ul style="list-style-type: none"> <li>• Type the name of the parent type in the <b>Parent type:</b> field.</li> <li>• Click <b>Select</b> to select a parent type. The <b>Artifact Selector</b> dialog appears. Select a parent type from the listbox.</li> <li>• Click the <b>Parent type:</b> link to create a parent type. The <b>New Documentum Artifact - Name and Location</b> wizard appears. For more information about creating an artifact, see <a href="#">Creating an artifact</a>, page 30.</li> </ul>
Relationship direction	<p>An integer specifying the relationship direction. The relationship direction can have the following values:</p> <ul style="list-style-type: none"> <li>• <b>From Parent to Child</b></li> <li>• <b>From Child to Parent</b></li> <li>• <b>Bidirectional</b></li> </ul> <p>The default relationship direction value is <b>From Parent to Child</b>.</p>

Property	Description
Permanent link	Determines whether the relationship is maintained when the parent is copied or versioned. Select this option to maintain the relationship between parent and child, in one of the following ways: <ul style="list-style-type: none"><li>• The child object is copied if the parent is copied.</li><li>• The child object is not copied.</li></ul>
Child-to-parent label	A string with up to 255 characters specifying a label for the child-to-parent relationship. Type the string in the text field and click <b>Add</b> .
Parent-to-child label	A string with up to 255 characters specifying a label for the parent-to-child relationship. Type the string in the text field and click <b>Add</b> .

# Managing Smart Containers

This chapter contains the following topics:

- [Smart containers, page 163](#)
- [Constructing a smart container, page 163](#)
- [Adding smart container elements, page 165](#)
- [Adding smart container relationships, page 170](#)

## Smart containers

Smart containers define objects and relationships in a template that is used to instantiate instances at runtime. Composer provides a smart container editor that lets developers construct smart containers declaratively instead of programmatically, thus greatly reducing the time to write DFC applications. After a smart container has been constructed the objects are similar to Documentum persistent objects.

Because a smart container template is intended to be used for repeated construction of a modeled composite object, each new instance of the composite object should be different. This is accomplished by parameterizing the smart container at design time.

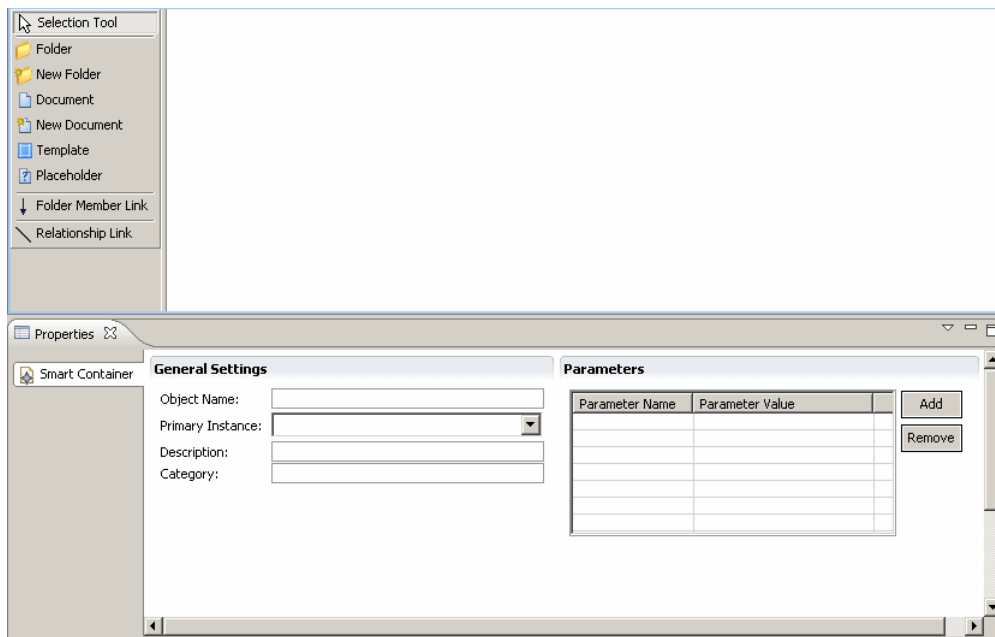
**Note:** When configuring the installation options for a smart container artifact, be sure to set the **Upgrade option** to **Create New Version Of Matching Objects**. Do not select **Overwrite Matching Objects** for smart container artifacts because overwriting smart container objects invalidates the model-instance association for existing instances.

## Constructing a smart container

Use the **Smart Container** editor to construct or modify a smart container.

**To construct a smart container:**

1. Open the **Select a wizard** dialog in one of the following ways:
  - From the Composer menu, select **File > New > Other**.
  - In your Composer project, expand the **Artifacts** folder and right-click **Types**. Select **New > Other**.
2. Select **Documentum > Smart Container**, then click **Next**.  
The **New Documentum Resource - Name and Location** dialog appears.
3. Enter the folder path and name of the project for which you want to construct a smart container in the **Folder:** field, or click **Browse** to select the project and folder path from a folder list.
4. Enter a file name for smart container in the **Artifact name:** field, then click **Next**.  
The **Smart Container** editor appears.



5. Configure the smart container properties, as described in [Table 43, page 164](#).

**Table 43. Smart container properties**

Parameter	Description
<b>General Settings</b>	
Object Name	A string specifying the name of the smart container. You can either accept the default name that is assigned by Composer or enter a new name.
Primary Instance	Specifies the primary object of the smart container. Select a primary instance from the drop-down list. Every smart container must have exactly one primary instance. The primary instance can be a new folder, existing folder, new document, existing document, or a template, but not a placeholder.

Parameter	Description
Description	A description of the smart container.
Category	A string specifying a discriminator that can be used in a filter, for example in drop-down lists.
<b>Parameters</b>	
Parameter Name	The name of a parameter that is used by the smart container at runtime.
Default Value	The value of the runtime parameter.

6. Add one or more artifacts to your smart container, as follows:
  - **Folder**, as described in [Adding a folder](#) , page 165
  - **New folder**, as described in [Adding a new folder](#) , page 166
  - **Document**, as described in [Adding a document](#) , page 167
  - **New document**, as described in [Adding a new document](#), page 168
  - **Template**, as described in [Adding a template](#) , page 168
  - **Placeholder**, as described in [Adding a placeholder](#), page 169
7. Add relationships to your artifacts, as described in [Adding smart container relationships](#), page 170.
8. Save your changes.
9. Configure the artifact installation parameters for the smart container, as described in [Configuring artifact install options](#), page 203.

**Note:** Be sure to set the upgrade option in the installation parameters to **Create New Version Of Matching Objects**. Do not select **Overwrite Matching Objects** for smart container artifacts because overwriting smart container objects invalidates the model-instance association for existing instances.


## Adding smart container elements

A smart container can contain various different elements, such as folders, documents, templates, and placeholders.

### Adding a folder

Use the **Folder** option in the smart container editor to add an instance of an existing folder to the smart container. Since you are adding a folder that exists in a repository, define the folder by adding either the folder's Documentum object id or path.

**To add a folder:**

1. Open the smart container editor, as described in [Constructing a smart container, page 163](#).
2. Select the  **Folder** icon and click the workspace. The folder appears in the smart container workspace.
3. Click the **Folder Info** tab in the **Properties** view and define the folder properties, as described in [Table 44, page 166](#).


**Table 44. Folder properties**

Parameter	Description
Display Name	The name of the folder that appears in the workspace. This name for display purposes only, so folders can be distinguished in the workspace. The display name is not used in a repository or any other application.
Object Id	The 16-character Documentum object id of the folder.
Path	The relative path for a location to which the folder is linked.

## Adding a new folder

A new smart container folder is like a regular new folder with the exception that a new smart container folder does not get instantiated until runtime.

**To add a new folder:**

1. Open the **Smart Container** editor, as described in [Constructing a smart container, page 163](#).
2. Select the  **New Folder** icon and click the workspace. A new folder appears in the smart container workspace.
3. Click the **New Folder Info** tab in the **Properties** view and define the folder properties, as described in [Table 45, page 166](#).

**Table 45. New folder properties**

Parameter	Description
Display Name	The name of the folder that appears in the workspace. This name for display purposes only, so folders can be distinguished in the workspace. The display name is not used in a repository or any other application.
Object Name	The object name of the new folder.


Parameter	Description
Type	The object type of the new folder. Click <b>Select</b> and select an object type from the list, or click the <b>Type</b> link to create an object type. For more information about types, see <a href="#">Chapter 18, Managing Types</a> .
Permission Set	The permission set assigned to the folder. Click <b>Select</b> and select a permission set from the list, or click the <b>Permission Set</b> link to create a permission set. For more information about permission sets, see <a href="#">Chapter 13, Managing Permission Sets (ACLs)</a> .

- Click the **Aspects Tab** to attach one or more aspects to the new folder. Click **Add** and select an aspect from the list or create an aspect. For more information about aspects, see [Chapter 7, Managing Aspects](#).
- Click the **Attributes Tab** to add one or more attributes to the new folder. Click **Add** and select an attribute from the list.
- Save your changes.

## Adding a document

Use the **Document** option in the smart container editor to add an instance of an existing document to the smart container. Since you are adding a document that exists in a repository, define the document by adding either the document's Documentum object id or path.

### To add a document:

- Open the **Smart Container** editor, as described in [Constructing a smart container, page 163](#).
- Select the  **Document** icon and click the workspace. The document appears in the smart container workspace.
- Click the **Instance Info** tab in the **Properties** view and define the document properties, as described in [Table 46, page 167](#).


**Table 46. Document instance properties**

Parameter	Description
Display Name	The name of the document instance that appears in the workspace. This name for display purposes only, so document instances can be distinguished in the workspace. The display name is not used in a repository or any other application.
Object Id	The 16-character Documentum object id of the document instance.
Path	The relative path for a location to which the document instance is linked.

## Adding a new document

A new smart container document is like a regular new document with the exception that a new smart container document does not get instantiated until runtime.

### To add a new document:

1. Open the **Smart Container** editor, as described in [Constructing a smart container, page 163](#).
2. Select the  **New Document** icon and click the workspace. A new document appears in the smart container workspace.
3. Click the **New Instance Info** tab in the **Properties** view and define the new document properties, as described in [Table 47, page 168](#).

**Table 47. New document instance properties**

Parameter	Description
Display Name	The name of the new document instance that appears in the workspace. This name for display purposes only, so new document instances can be distinguished in the workspace. The display name is not used in a repository or any other application.
Object Name	The object name of the new document instance.
Type	The object type of the new document instance. Click <b>Select</b> and select an object type from the list, or click the <b>Type</b> link to create an object type. For more information about types, see <a href="#">Chapter 18, Managing Types</a> .
Permission Set	The permission set assigned to the document instance. Click <b>Select</b> and select a permission set from the list, or click the <b>Permission Set</b> link to create a permission set. For more information about permission sets, see <a href="#">Chapter 13, Managing Permission Sets (ACLs)</a> .

4. Click the **Aspects** tab to attach one or more aspects to the new document instance. Click **Add** and select an aspect from the list or create an aspect. For more information about aspects, see [Chapter 7, Managing Aspects](#).
5. Click the **Attributes** tab to add one or more attributes to the new document instance. Click **Add** and select an attribute from the list.
6. Save your changes.


## Adding a template

A smart container template is an existing document that you want to have copied into your smart container at construction.

### To add a template:

1. Open the **Smart Container** editor, as described in [Constructing a smart container, page 163](#).



2. Select the  **Template** icon and click the workspace. The template appears in the smart container workspace.
3. Click the **Template Info** tab in the **Properties** view and define the template properties, as described in [Table 48, page 169](#).

**Table 48. Template properties**


Parameter	Description
Display Name	The name of the template that appears in the workspace. This name for display purposes only, so new document instances can be distinguished in the workspace. The display name is not used in a repository or any other application.
Object Name	The object name of the template.
Permission Set	The permission set assigned to the template. Click <b>Select</b> and select a permission set from the list, or click the <b>Permission Set</b> link to create a permission set. For more information about permission sets, see <a href="#">Chapter 13, Managing Permission Sets (ACLs)</a> .
Based on	
Object Id	The 16-character Documentum object id of the template.
Path	The relative path for a location to which the template is linked.

4. Click the **Aspects** tab to attach one or more aspects to the template. Click **Add** and select an aspect from the list or create an aspect. For more information about aspects, see [Chapter 7, Managing Aspects](#).
5. Click the **Attributes** tab to add one or more attributes to the template. Click **Add** and select an attribute from the list.
6. Save your changes.

## Adding a placeholder

A placeholder object is similar to a template object, but a placeholder object is not created at construction time. A placeholder object lets a modeler indicate that other objects must be added later in order for the composite object to be considered “complete”. A placeholder must be a “leaf” node.

### To add a placeholder:

1. Open the **Smart Container** editor, as described in [Constructing a smart container, page 163](#).
2. Select the  **Placeholder** icon and click the workspace. The placeholder appears in the smart container workspace.
3. Click the **Placeholder Info** tab in the **Properties** view and define the placeholder properties, as described in [Table 49, page 170](#).

**Table 49. Placeholder properties**

Parameter	Description
Display Name	The name of the placeholder that appears in the workspace. This name for display purposes only, so new placeholders can be distinguished in the workspace. The display name is not used in a repository or any other application.
Object Name	The object name of the placeholder.
Type	The object type of the placeholder. Click <b>Select</b> and select an object type from the list, or click the Type link to create an object type. For more information about types, see <a href="#">Chapter 18, Managing Types</a> .
Required	Specifies whether the placeholder must be assigned an object type.

4. Save your changes.

## Adding smart container relationships

Relationships between smart container objects can take the form of folder links and generic relationships. The smart container editor provides two options to visually distinguish a relationship:

- **Folder Member Link**

You can use the **Folder Member Link** relation when you are modeling folders and their members.

- **Relation**

You can use the **Relation** option for all other generic relationships. For example, a relationship between an insurance claim and a customer.

### To add a relationship:

1. Click the **Folder Member Link** tab or **Relation** tab to activate it.
2. Click the first smart container object.
3. Click the second smart container object.  
An arrow appears indicating that the two objects are now connected.

## Managing SysObjects

This chapter contains the following topics:

- [SysObjects, page 171](#)
- [Creating a SysObject, page 171](#)
- [Viewing and modifying SysObject attributes, page 173](#)

## SysObjects

The SysObject type is the parent type of the most commonly used objects in the Documentum system. SysObjects are the supertype, directly or indirectly, of all object types in the hierarchy that can have content. The SysObject type's defined attributes store information about the object's version, the content file associated with the object, the security permissions on the object and other information important for managing content. The SysObject subtype most commonly associated with content is dm\_document.

## Creating a SysObject

Use the SysObject editor to create or modify a SysObject.

### To create a SysObject:

1. Open the **Select a wizard** dialog in one of the following ways:
  - From the Composer menu, select **File > New > Other**.
  - In your Composer project, expand the **Artifacts** folder and right-click **SysObjects**. Select **New > Other**.

The **Select a wizard** dialog appears.

2. Select **Documentum > SysObject**, then click **Next**.

The **New Documentum Resource - Name and Location** dialog appears.

3. Enter the folder path and name of the project for which you want to create a SysObject in the **Folder:** field, or click **Browse** to select the project from a folder list.
4. Enter a file name for the SysObject in the **Artifact name:** field, then click **Next**.

The **Type, File, and Format** dialog appears.

5. Enter the type, file, and format information for the SysObject, if applicable, as described in [Table 50, page 172](#), then click **Next**.

**Note:** If you do not want to enter a file name and format for the SysObject, click **Finish**.

**Table 50. SysObject properties**

Property	Description
Type	The object type that contains the content. By default, the object type is set to dm_sysobject. Click <b>Select</b> to select a different object type from the drop-down list.
File	The name of the file that contains the content, if applicable. Click <b>Browse</b> to select the file from your local machine or network drive.
Format	The format of the content file. Click <b>Select</b> to select a file format from the drop-down list.

The **SysObject** editor appears.

6. Click **Add** in the **Attached Aspects** section to attach one or more aspects to the SysObject. The **Aspect Module Artifact** dialog appears.
7. Select an aspect from the list or click **New** to create an aspect. For more information about aspects, see [Chapter 7, Managing Aspects](#).

## Viewing and modifying SysObject attributes

The **Attributes** tab in the **SysObject** editor lets you view attributes and modify the attribute values associated with the specified SysObject.

**To view or modify SysObject attributes:**

1. Click the **Attributes** tab in the **SysObject** editor.  
The **Attributes** view appears.
2. Select an attribute from the list to view the attribute details.

## Attributes

List

☒ Show editable attributes only

☐ Show all attributes

Name	Value	Origin
title		dm_sysobject
subject		dm_sysobject
authors		dm_sysobject
keywords		dm_sysobject
a_is_hidden	false	dm_sysobject
a_application_type		dm_sysobject
a_compound_architecture		dm_sysobject
resolution_label		dm_sysobject
a_special_app		dm_sysobject
language_code		dm_sysobject
ad_name		dm_sysobject
a_storage_type		dm_sysobject

Details

Attribute Name: title

Attribute Value:

General

Attributes

In the **Attributes** view, you can do the following:

- Select the **Show editable attributes only** radio button to list only attributes that can be modified.
  - Select the **Show all attributes** radio button to list all attributes for the SysObject.
  - Enter or modify the value of an editable attribute in **Attribute Value** field of the **Details** section.
3. Save your changes.



## Managing Types

This chapter contains the following topics:

- [Object types , page 175](#)
- [Creating a standard object type, page 176](#)
- [Creating a lightweight object type, page 179](#)
- [Configuring constraint expressions, page 183](#)
- [Adding, deleting, or modifying events, page 184](#)
- [Adding type attributes, page 184](#)
- [Configuring the type UI information , page 192](#)

### Object types

An object type is like a template and represents a class of objects. Composer lets you create two types:

- Standard objects
- Lightweight objects

Every object type is defined by a set of attributes. When an object is created, its attributes are set to values that describe that particular instance of the object type. For example, two attributes of the document object type are title and subject. When users create a document, they provide values for the title and subject attributes that are specific to that document. For more information about objects and object types, see *Documentum Content Server Fundamentals*.

Lightweight objects are part of an object model enhancement introduced to share system managed metadata among objects which only hold application-specific data. For example, policies for security, retention, and storage, are stored in a regular system object that is shared among all the lightweight objects. Because the system managed metadata is stored only once, it significantly reduces the disk storage requirements and improves the ingestion performance.

**Note:** Currently only applications designed for Documentum High-Volume Server can make proper use of lightweight objects. Documentum High-Volume Server is an extension of Documentum Content Server that supports features implemented to solve common problems with large content stores, such as email archiving. It requires an additional license key specified when Content Server is installed. For more information about lightweight object types and Documentum High-Volume Server, see *EMC Documentum High-Volume Server Developer Guide*.

# Creating a standard object type

Use the **Type** editor to create or modify a standard object type.

## To create a standard object type:

1. Open the **Select a wizard** dialog in one of the following ways:
  - From the Composer menu, select **File > New > Other**.
  - In your Composer project, expand the **Artifacts** folder and right-click **Types**. Select **New > Other**.

The **Select a wizard** dialog appears.

2. Select **Documentum > Documentum Artifact > Type**, then click **Next**.

The **New Documentum Artifact - Name and Location** dialog appears.

3. Enter the folder path and name of the project for which you want to create an object type in the **Folder:** field, or click **Browse** to select the project from a folder list.
4. Enter a file name for the object type in the **Artifact name:** field
5. Select **Standard object type**, then click **Next**.

The **Type** editor appears with the **General** tab selected.

6. Enter the object type information in the **Info**, **Default Attached Aspects**, **Constraints**, and **Events** sections, as described in [Table 51](#), [page 176](#).

**Table 51. Type information on General tab**

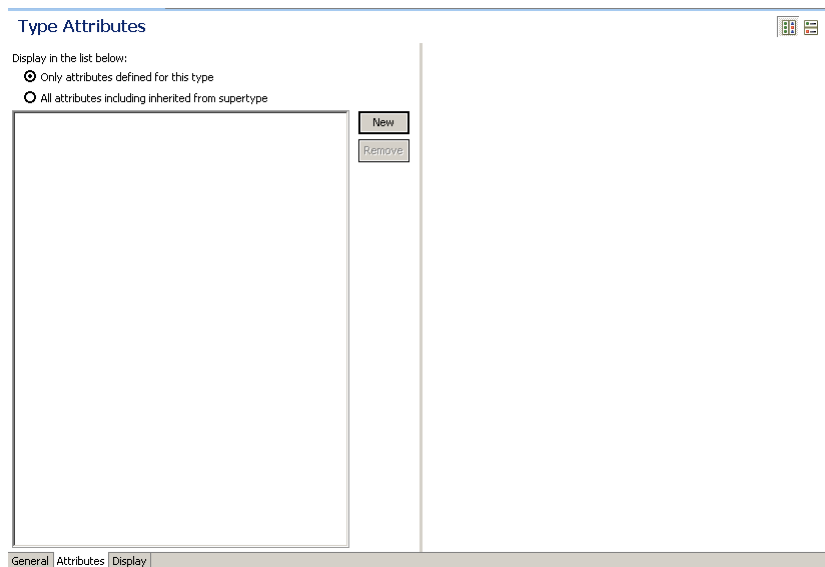
Property	Description
<b>General</b>	



Property	Description
Type name	<p>A string with specifying the name of the type. The following rules apply to all type names:</p> <ul style="list-style-type: none"> <li>• A maximum of 27 characters, all lower-case. The Content Server is case-insensitive and stores all type names in lower-case.</li> <li>• The first character must be a letter, the remaining characters can be letters, digits, or underscores</li> <li>• Cannot contain any spaces or punctuation</li> <li>• Cannot end in an underscore (_) </li> </ul>
Is Shareable	Select this option if you want the properties of this type to be shareable with other object types.
Supertype	The supertype of the new type. A supertype is a type that is the basis for another type. The new type inherits all the properties of the specified supertype. Click <b>Select ...</b> and select a supertype from the listbox.
Storage area	Specifies the default storage location for instances of this type. If you do not assign a custom default storage location, Composer automatically assigns the system default storage location.
<b>Default Attached Aspects</b>	Click <b>Select ...</b> to specify one or more aspects that are attached to instances of this type. Aspects let you modify the behavior of type instances. For more information about attaching aspects, see <a href="#">Attaching aspects, page 179</a> .
<b>Constraints</b>	Constraints are internal consistency requirements in the form of Docbasic expressions that relate the types attribute values to one another or to constant values.
Expression	The Docbasic expression defining the constraint. Click <b>New</b> to create a new expression. For more information about creating or modifying an expression, see <a href="#">Configuring constraint expressions, page 183</a> .
Enforcement	<p>Specifies whether applications should enforce this constraint or not. Click the table cell in the Enforcement column to enable or disable constraint enforcement for the associated expression. The enforcement field can have two values, as follows:</p> <ul style="list-style-type: none"> <li>• <b>disabled</b>: The constraint is disabled</li> <li>• <b>ApplicationEnforced</b>: The constraint is enforced by the applications that use this type.</li> </ul>

Property	Description
<b>Events</b>	Events are specific actions on objects. You can only create and modify application events, not system events. Click <b>New</b> to enter a new event. To edit or remove an event, select the event and click <b>Edit</b> or <b>Remove</b> , respectively. For more information about creating an event, see <a href="#">Adding, deleting, or modifying events, page 184</a> .
Only events defined for this type	Select this option to display events that are defined for this type in the events table.
All events including inherited from supertype	Select this option to display all events for this type in the events table, including events that are inherited from the supertype.
Event name	A string specifying the name of the event that is associated with instances of this type.
Event label	A string that specifies the label for the event.

- Click the **Attributes** tab.  
The **Type Attributes** view appears.



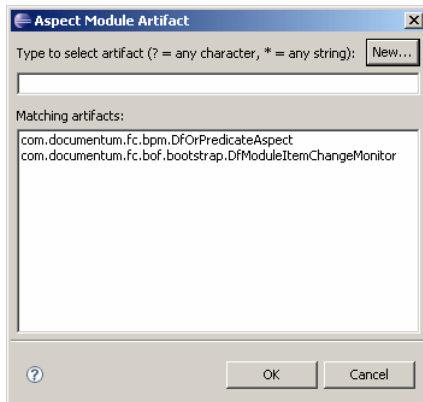
- Click **Add** to add a new type attribute. For more information about adding type attributes, see [Adding type attributes, page 184](#).
- Click the **Display** tab.  
The **Type UI Information** view appears ([Figure 13, page 192](#)).
- Enter the type UI information in the **Display Configuration** and **Application Interface Display** sections, as described in [Configuring the type UI information , page 192](#).

## Attaching aspects

When you create an object type, you can also assign aspects that are attached to the object type by default.

### To attach an aspect:

1. Click **Add** in the **Default Attached Aspects** section of the **General** tab in the type editor. The **Aspect Module Artifact** dialog appears.



2. Select an aspect from the **Matching Artifacts** listbox, then click **OK**.

**Note:** If there are no aspects listed in the **Matching Artifacts** listbox, no aspects have been created yet. For more information about creating an aspect, see [Creating an aspect type, page 87](#).

## Creating a lightweight object type

Before you create a lightweight object type, verify that the application you are building can actually utilize this type of object. Currently only archiving applications designed for Documentum High-Volume Server have a use for lightweight object types. Documentum High-Volume Server is an extension of Documentum Content Server that supports features implemented to solve common problems with large content stores, such as email archiving. It requires an additional license key specified when Content Server is installed.

### To create a lightweight object type:

1. Open the **Select a wizard** dialog in one of the following ways:
  - From the Composer menu, select **File > New > Other**.
  - In your Composer project, expand the **Artifacts** folder and right-click **Types**. Select **New > Other**.

The **Select a wizard** dialog appears.

2. Select **Documentum > Documentum Artifact > Type**, then click **Next**.

The **New Documentum Artifact - Name and Location** dialog appears.

- Enter the folder path and name of the project for which you want to create an object type in the **Folder:** field, or click **Browse** to select the project from a folder list.
- Enter a file name for the object type in the **Artifact name:** field
- Select **Lightweight object type**, then click **Next**.

The **Lightweight Type** editor appears with the **General** tab selected.

- Enter the object type information in the **Info**, **Constraints**, and **Events** sections, as described in [Table 52, page 180](#).

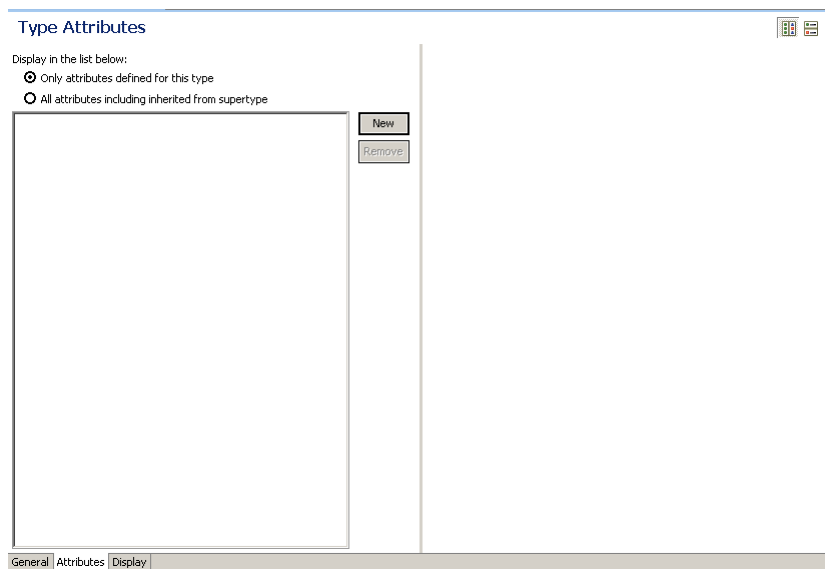
**Table 52. Lightweight type information on General tab**

Property	Description
<b>General</b>	
Type name	<p>A string with specifying the name of this lightweight object type. The following rules apply to all type names:</p> <ul style="list-style-type: none"> <li>A maximum of 27 characters, all lower-case. The Content Server is case-insensitive and stores all type names in lower-case.</li> <li>The first character must be a letter, the remaining characters can be letters, digits, or underscores</li> <li>Cannot contain any spaces or punctuation</li> <li>Cannot end in an underscore (_)</li> </ul>
Shared Parent	The parent object type that is sharing its properties with this lightweight object.

Property	Description
Supertype	The supertype of this lightweight object type. A supertype is a type that is the basis for another type. The new type inherits all the properties of the specified supertype. Click <b>Select ...</b> and select a supertype from the listbox.
Materialization Behavior	<p>Specifies when this lightweight object type shares a parent object or has its own private copy of a parent. A lightweight object is classified as unmaterialized when it shares a parent object with other lightweight objects. A lightweight object is classified as materialized, when a lightweight object has its own private copy of a parent. Materializing a lightweight object might drastically increase the size of database tables.</p> <p>The materialization behavior be set to the following values:</p> <ul style="list-style-type: none"> <li>• <b>Auto-Materialize:</b> The lightweight object is materialized automatically when certain operations occur, for example a checkout/checkin operation or a branch operation.</li> <li>• <b>On-Request:</b> The lightweight object is materialized only when requested by an explicit API call. For more information on the APIs that can be used to materialize a lightweight object, see the <i>EMC Documentum Archive Developer Guide</i>.</li> <li>• <b>Disallow:</b> The lightweight object is never materialized.</li> </ul> <p>By default the materialization behavior is set to <b>Auto-Materialize</b>.</p>
Constraints	Constraints are internal consistency requirements in the form of Docbasic expressions that relate the types attribute values to one another or to constant values.
Expression	The Docbasic expression defining the constraint. Click <b>New</b> to create an expression. For more information about creating or modifying an expression, see <a href="#">Configuring constraint expressions, page 183</a> .

Property	Description
Enforcement	<p>Specifies whether applications should enforce this constraint or not. Click the table cell in the Enforcement column to enable or disable constraint enforcement for the associated expression. The enforcement field can have two values, as follows:</p> <ul style="list-style-type: none"> <li>• <b>disabled:</b> The constraint is disabled</li> <li>• <b>ApplicationEnforced:</b> The constraint is enforced by the applications that use this type.</li> </ul>
Events	<p>Events are specific actions on objects. You can only create and modify application events, not system events. Click <b>New</b> to enter a new event. To edit or remove an event, select the event and click <b>Edit</b> or <b>Remove</b>, respectively. For more information about creating an event, see <a href="#">Adding, deleting, or modifying events, page 184</a>.</p>
Only events defined for this type	Select this option to display events that are defined for this type in the events table.
All events including inherited from supertype	Select this option to display all events for this type in the events table, including events that are inherited from the supertype.
Event name	A string specifying the name of the event that is associated with instances of this type.
Event label	A string that specifies the label for the event.

7. Click the **Attributes** tab.  
The **Type Attributes** view appears.



8. Click **Add** to add a new type attribute. For more information about adding type attributes, see [Adding type attributes, page 184](#).
9. Click the **Display** tab.  
The **Type UI Information** view appears ([Figure 13, page 192](#)).
10. Enter the type UI information in the Display Configuration and Application Interface Display sections, as described in [Configuring the type UI information , page 192](#).

## Configuring constraint expressions

Constraints are internal consistency requirements in the form of Docbasic expressions that relate the types attribute values to one another or to constant values.

### To add a constraint expression for a type:

1. Click **Add** in the **Constraints** section of the **Type Overview** tab in the type editor.

The **Edit Constraint** dialog appears.



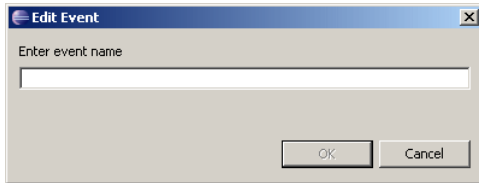
2. Type a valid Docbasic constraint expression that resolves to true or false in the **Expression:** text box. The Docbasic expression resolves to true when the constraint is fulfilled and false when the constraint is violated.  
The **Reset** button returns the contents of the **Constraint Expression** text box to the last value that passed a syntax test.
3. Type a message for applications to display when the constraint is violated in the **Error message when constraint is violated:** text box.
4. Check the **Enforce Constraint** checkbox to instruct applications to enforce this constraint or leave it unchecked to not enforce the constraint.
5. Click **OK** to save your changes.

## Adding, deleting, or modifying events

Events are specific actions on objects. You can only create and modify application events, not system events.

### To create an event:

1. Click **New** in the **Events** section of the **Type Overview** tab in the type editor.  
The **Edit Event** dialog appears.



2. Enter a name for the event, then click **OK**. The event appears in the event table.

## Adding type attributes

Type attributes are configured in the **Attributes** tab of the type editor. A type attribute is a property that applies to all objects of that type. When an object is created, its attributes are set to values that describe that particular instance of the object type.

### To create an attribute:

1. Click the **Attributes** tab in the type editor to display the **Attributes** view.
2. Click **New** to create an attribute entry, then click the + sign to display the configuration options.  
The **Type Attributes** view expands and appears the **Structure** and **Constraints** section.



**Type Attributes**

Display in the list below:

- ☒ Only attributes defined for this type
- ☐ All attributes including inherited from supertype

**newattribute1**

- Application Interface Display
- Value mapping

**Structure**

Name: newattribute1

Data type: STRING

Length: 10

Repeating: ☐ Non-qualifiable: ☐

Default values:


**Constraints**

Expression	Enforcement	New...

☐ Ignore supertype constraints when executing application

☐ Attribute cannot be blank

☐ Attribute is read-only

☒ Attribute can have NULL value

☐ Attribute can be modified on immutable objects

General Attributes Display

- Configure the attribute structure, as described in [Configuring the attribute structure, page 185](#).
- Configure the attribute constraints, as described in [Configuring attribute constraints, page 186](#).
- Click the UI icon in the **Attribute** pane to display the attribute's UI configuration options.
- Configure the attribute's UI options, as described in [Configuring the type attribute UI, page 188](#).
- Click the Value mapping icon to display the attribute's value mapping options.
- Configure the attribute conditions, as described in [Configuring conditional attribute values, page 190](#).
- Configure the attribute value mapping, as described in [Configuring attribute value mapping, page 191](#).

## Configuring the attribute structure

The attribute structure is configured in the **Structure** section of the **Type Attributes** view ([Figure 8, page 185](#)).

**Figure 8. Structure section in Type Attributes view**

**Structure**

Name: NewAttribute1

Data type: STRING

Length: 0

Repeating: ☐ Non-qualifiable: ☐

Default values:


New Remove

Enter the attribute structure properties, as described in [Table 53, page 186](#).

**Table 53. Attribute structure properties**

Property	Description
Name	A string specifying the name of the new attribute. The attribute name must use all lowercase letters, cannot begin with dm_, a_, i_, r_, a numeral, space, or single quote, and cannot be named select, from, or where.
Data type	The data type of the new attribute. Select one of the following data types from the drop-down list: <ul style="list-style-type: none"><li>• <b>BOOLEAN</b></li><li>• <b>INTEGER</b></li><li>• <b>STRING</b></li><li>• <b>ID</b></li><li>• <b>TIME</b></li><li>• <b>DOUBLE</b></li><li>• <b>UNDEFINED</b></li></ul>
Length	This parameter only applies to attributes that use the <b>STRING</b> data type. Enter the number of characters for this attribute. The maximum number of characters that you can assign to this attribute depends on the database where you are installing the application.
Repeating	Specifies whether this attribute can have more than one value. Select the checkbox to allow more than one value for this attribute.
Non-qualified	Specifies whether this attribute is qualifiable or non-qualifiable. The properties and values of a non-qualifiable attribute are stored in a serialized format and do not have their own columns in the underlying database tables that represent the object types for which they are defined. Consequently, non-qualifiable attributes cannot be used in queries because they are not exposed in the database.
Default values	Lets you specify one default value for a single-value attribute or multiple default values for a repeating attribute. Click <b>New</b> to enter a default value.

## Configuring attribute constraints

Attribute constraints are configured in the **Constraint** section of the **Type Attributes** view ([Figure 9, page 187](#)). Constraints are internal consistency requirements in the form of Docbasic expressions that relate the types attribute values to one another or to constant values.

Figure 9. Attribute constraints

**Constraints**

Expression	Enforcement

☐ Ignore supertype constraints when executing application  
☐ Attribute cannot be blank  
☐ Attribute is read-only  
☒ Attribute can have NULL value  
☐ Attribute can be modified on immutable objects

Enter or specify the attribute constraint properties, as described in [Table 54, page 187](#).

Table 54. Attribute constraint properties

Property	Description
Expression	The Docbasic expression defining the constraint. Click <b>New</b> to create an expression. For more information about creating or modifying an expression, see <a href="#">Configuring constraint expressions, page 183</a> .
Enforcement	Specifies whether applications should enforce this constraint or not. Click the table cell in the Enforcement column to enable or disable constraint enforcement for the associated expression. The enforcement field can have two values, as follows: <ul style="list-style-type: none"> <li>• <b>disabled</b>: The constraint is disabled</li> <li>• <b>ApplicationEnforced</b>: The constraint is enforced by the applications that use this type.</li> </ul>
Ignore supertype constraints when executing application	Select this option to specify that applications should ignore supertype constraints for attributes of this type.
Attribute cannot be blank	Select this option to specify that the attribute must have a value. End users must enter a value for this attribute when the application executes.
Attribute is read-only	Select this option to specify that the attribute is read-only. End users cannot change the value of the attribute when the application executes.
Attribute can have NULL value	Select this option to specify that the attribute does not need to have a value assigned. End users do not need to enter a value for this attribute when the application executes.
Attribute can be modified on immutable objects	Select this option if you want users to be able to modify the attribute even though the object itself is immutable (unchangeable).

## Configuring the type attribute UI

The attribute UI specifies how the attribute is displayed in client applications and is configured in the **UI** view. To open the **UI** view, click the  UI icon in the attribute directory tree.

The **UI** view appears ([Figure 10, page 188](#)).

**Figure 10. Type attribute UI view**



The screenshot shows the 'General' and 'Search' sections of the Type attribute UI view. The 'General' section includes fields for 'Label', 'Input mask', 'Category', 'User help', and 'Comment for developers'. The 'Search' section includes a checkbox for 'Is searchable', a list of 'Available operators' (equals, notequal, greaterthan, lessthan, greaterequal, lessthanequal, beginswith, contains, doesnotcontain, endswith, in, notin, between, isnull, notnull, not, null), and buttons for 'Add >' and '< Remove'. Below these are fields for 'Default search value' and 'Default search operator'.

Enter or specify the attribute UI properties in the **General** and **Search** sections, as described in [Table 55, page 188](#).

**Table 55. Type attribute UI properties**

Property	Description
<b>General</b>	
Label	The name that is displayed for this attribute in the client application.
Input mask	<p>Specifies the characters and format that an end user can enter for this attribute using the client application. An input mask consists of mask characters and literals. A backslash (\) converts the character following it to a literal. The input mask can have the following values:</p> <ul style="list-style-type: none"> <li>#—Numeric characters (0-9).</li> <li>A—Alphanumeric characters (0-9, a-z, A-Z)</li> <li>&amp;—Any ASCII character</li> <li>?—Alphabetic characters (a-z, A-Z)</li> <li>U—Alphabetic, gets converted to uppercase</li> </ul>

Property	Description
	<ul style="list-style-type: none"> <li>• <b>L</b>—Alphabetic, gets converted to lowercase</li> </ul> <p>For input mask examples, see <a href="#">Table 56, page 189</a>.</p>
Category	A string that specifies a custom tab that is displayed in Desktop client. The value entered in the category field is used unless the inheritance with the parent types display configuration is broken or a display configuration is already specified for either the type or its parent.
User help	Optional description for the type that is displayed in the application.
Comment for developers	Optional comments for developers.
<b>Search</b>	
Is searchable	Specifies whether an attribute is searchable. Select this option to enable clients to allow users to search a repository for attribute values in objects, if the objects are derived from the attribute's type.
Available operators	Lists the operators that can be specified for an attribute search. Select one or more operators and click <b>Add</b> to move the operators to the <b>Allows operators</b> column.
Allowed operators	Specifies the search operators than a client application can use to search for an attribute value.
Default search value	The search value that clients display by default. Optional parameter. If no default search value is specified, the client displays a blank field.
Default search operator	The search operator that clients display by default. Optional parameter. If no default search operator is specified, the client displays a blank field.

[Table 56, page 189](#) shows a mask value and an example of valid user input.

**Table 56. Input mask examples**

Mask Value	Description	Valid User Input
##/##/####	Specifies a date entry.	08/23/1968
##:## UU	Specifies a time entry.	2:42 PM
###-##-####	Specifies a format for entering a numerical sequence, such a US social security number.	111223333
???????????????	Specifies a sequence for entering up to 15 letters, for example a city name.	Munich

## Configuring conditional attribute values

Conditional value mapping provides a list of values that a client program displays at runtime for an object attribute. A user can select a value from this list or add a new one to it. There are two kinds of conditional value mappings:

- **Default value mapping**—Values that are displayed when no value is selected, for example when a new object is created.
- **Conditional value mapping**—Values are displayed when a specified condition is satisfied, for example, when the values displayed in one attribute depend on the value of another attribute.

Conditional value mappings are configured in the **Conditional Assistance** section of the **Value Mapping** view (Figure 11, page 190).

**Figure 11. Conditional assistance view**

### To create a conditional value mapping:

1. Click **New**.

The **Conditional Value Assistance** dialog appears.

2. Specify the conditional value properties, as described in Table 57, page 190.

**Table 57. Conditional value properties**

Property	Description
Expression	A Docbasic expression that specifies the condition. The Docbasic expression must resolve to true or false.

Property	Description
Fixed list	Specifies that the values are associated with the condition in form of a fixed list. Select this option if you want to use a fixed list of values and enter the respective values in the <b>Values</b> field.
Values	Specifies the values that are associated with the condition. Type the value in the <b>Value</b> field, separated by a comma.
List is complete	Specifies that the user cannot add any more values to the list.
Query	Specifies that the values that are associated with the condition are obtained by a query. Enter the query in the <b>Query</b> textbox. You can use the \$value keyword to resolve attribute values at runtime.
Allow cached queries	Select this option to allow cached queries.
Query attribute providing data	Specifies the query attribute that contains the data value for the condition. Enter the name of the query attribute.

- Click **OK** to save your changes.

## Configuring attribute value mapping

Value mapping associates or maps attribute values to strings that are displayed in a client program. When a mapped string, which is displayed in the client program, is selected and the object is saved to the repository, the corresponding value is saved to the mapped string. Attribute values are mapped in the **Value Mapping Table** section of the **Value Mapping** view ([Figure 12, page 191](#)).

**Figure 12. Value mapping table**

**Value Mapping Table**

Enter the data value, the display string to be shown in the UI, and an optional description:

Attribute Value	Display String	Description

New Remove

### To create or modify a value mapping:

- Click **New** to create a dataValue/displayValue pair in the **Attribute Value** and the **Display String** columns.
- Click the **dataValue** field in the **Attribute Value** column and enter the attribute value that is saved in the repository.
- Click the **displayValue** field in the **Display String** column and enter the string that is displayed as the mapped value in the client application.
- Click the field in the **Description** column to enter a string that describes the value mapping.

To remove a value mapping, select the corresponding row in the **Value Mapping Table** and click **Remove**.

## Configuring the type UI information

The type UI information view lets you specify which attributes are displayed in Documentum clients and custom applications. Click the **Type UI** tab in the type editor to display the **Type UI Information** view (Figure 13, page 192).

**Figure 13. Type UI information view**

### To configure one or more attributes to be displayed in clients:

1. Enter the type UI information as described in [Table 58, page 192](#).

**Table 58. Type UI information**

Property	Description
<b>Display Configuration</b>	
Scope	The name of the application, in which the type attribute is displayed. The name of the application must exist in the repository.
Display configuration list	<p>Specifies the tab on which the attribute is displayed. You can add, remove, rename, and change the position of a tab, as follows:</p> <ul style="list-style-type: none"> <li>• Click <b>New</b> to add a new tab. The Display Configuration dialog appears. For more information about adding a tab to display an attribute in a client application, see <a href="#">Adding a tab , page 193</a>.</li> <li>• To remove a tab, select the tab name in the list, then click <b>Remove</b>.</li> <li>• To rename a tab, select the tab name in the list, then click <b>Rename</b>.</li> </ul>



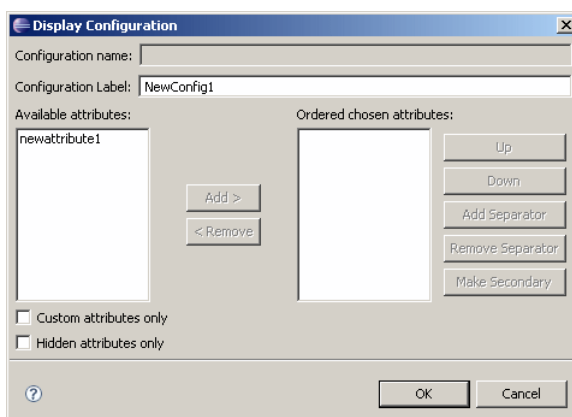
Property	Description
Attributes in display configuration	<ul style="list-style-type: none"> <li>To change the order in which the tabs are displayed, select the tab name in the list, then click <b>Up</b> or <b>Down</b> to move the tab to the desired position.</li> </ul> <p>Lets you modify the attributes that are displayed on a tab.</p>
<b>Application Interface Display</b>	
Type label	A string that the client application displays for this type.
User help	Optional description for the type that is displayed in the application.
Comments for developers	Optional comments for developers.

## Adding a tab

Use the **Display Configuration** dialog to add a tab.

### To add a tab:

- Click **Add** in the **Display Configuration List** section of the **Type UI Information** view ([Figure 13, page 192](#)). A default tab (**NewConfig1**) for the new tab appears in the **Display Configuration List** textbox.
- Select the default tab and then click **Edit**.  
The **Display Configuration** dialog appears.



- Configure the tab properties, as described in [Table 59, page 194](#).

**Table 59. Tab configuration properties**

Tab properties	Description
Configuration name	A string that specifies the tab name. You can either enter a new tab name or accept the default tab name. The configuration name is displayed in the client application.
Available attributes	Shows a list of the attributes that can be displayed on the tab. Select the attribute that you want to display on the tab and click <b>Add</b> . The attribute appears in the <b>Ordered chosen attributes</b> list. If the available attributes list is empty, no attributes have been configured yet. For more information about configuring attributes, see <a href="#">Adding type attributes, page 184</a> .
Ordered chosen attributes	<p>Specifies which attributes are displayed on the tab and how they are displayed. You can arrange how the attributes are displayed on the tab by selecting the attribute and using the following buttons:</p> <ul style="list-style-type: none"><li>• <b>Up</b>—Moves the attribute up in the display order.</li><li>• <b>Down</b>—Move the attribute down in the display order.</li><li>• <b>Add Separator</b>—Adds a separator between the selected and the following attribute.</li><li>• <b>Remove Separator</b>—Removes the separator.</li><li>• <b>Make Secondary</b>—Force attributes to be displayed on a secondary page, if not all attributes can fit on one tab.</li></ul>
Custom attributes only	Select this option to display only custom attributes in the <b>Available attributes</b> list.
Hidden attributes only	Select this option to display only hidden attributes in the <b>Available attributes</b> list.

4. Click **OK** to save your changes.

# Managing XML Applications

This chapter contains the following topics:

- [Understanding XML applications and the application configuration file, page 195](#)
- [Creating an XML Application artifact, page 195](#)
- [Viewing or modifying an XML application configuration file, page 196](#)

## Understanding XML applications and the application configuration file

XML applications customize and automate how XML objects and linked unparsed entities are stored in a repository. XML applications can be configured to automatically recognize different types of XML documents and set up rules to determine where they are stored, whether they should be divided into smaller chunks, how to extract and assign metadata to an object in the repository, what level of security to assign, and whether to attach a document lifecycle.

## Creating an XML Application artifact

Composer has a built-in editor that allows you to define XML Application artifacts.

To create an XML Application artifact:

1. Right-click the **XML Applications** folder of your project in the **Documentum Navigator** view and select **New > Other**.
2. Select **Documentum Artifact > XML Application** and click **Next**. The **New XML Application Artifact Wizard** appears.
3. In the **Artifact name** field, enter the name that you want to give to the XML Application artifact.
4. If you have an existing XML configuration file that you want to use, select the **Use an existing configuration file on the file system** radio button and click **Browse** to find the file. Otherwise select the **Use Documentum's default configuration file as a template** radio button.
5. Click **Finish**. The XML Application editor appears.
6. In the Root Elements section, add any root elements that you want to process:

- a. Select a root element from the list or enter one in the **Or type a new root element to add:** field and click the **Add >** button.
7. In the **Advanced** tab, specify a DTD or Schema to validate XML applications if you want to use one:
  - a. Select the **Use Grammar** checkbox.
  - b. Select the **Repository** radio button if you want to manage the DTD or Schema in the repository. Select **Local File System** if you want to manage the DTD or Schema on your local file system.
  - c. Select the **XML Schema** or **DTD** radio button depending on what you want to use and specify the XML Schema or DTD. If you are managing the XML Schema or DTD in the repository, the artifact must be in your project so you can select it.
8. In the Supporting Documents section, add any artifacts that are needed for your XML application.
9. In the Sample Documents section, add any sample XML documents.

## Viewing or modifying an XML application configuration file

Composer lets you import an existing XML application into a project. You can then modify the XML application configuration file using the **XML Configuration File** editor. The **XML Configuration File** editor provides lists of rules organized into types and allows you to base rules on elements in your XML document. There are four different types of rules:

- **XML content rule**

The XML content rule is the most frequently used rule and applies to parsed XML content. The main function of an XML Content Rule is usually to chunk XML content in the document, but it can have other primary purposes, such as assigning metadata to an XML document that is not chunked.

- **Link rule**

A link rule uses links in the XML document to external files or references to NDATA entities to locate and handle unparsed entities, such as graphics files. The Link Rule works like an XML Content Rule in terms of assigning an object location, metadata values, permissions, and so on. You can also specify whether the linked file should be treated as a child or a peer of the XML virtual document, and whether this is a permanent link.

- **Non XML content rule**

A non-XML content rule specifies how the server handles non-XML content contained in the XML document, such as base64-encoded data. A non-XML content rule works like XML content rules and link rules in terms of assigning an object location, metadata values, or permissions. In addition, you can specify the file format of the repository object that contains the decoded data.

- **Element rule**

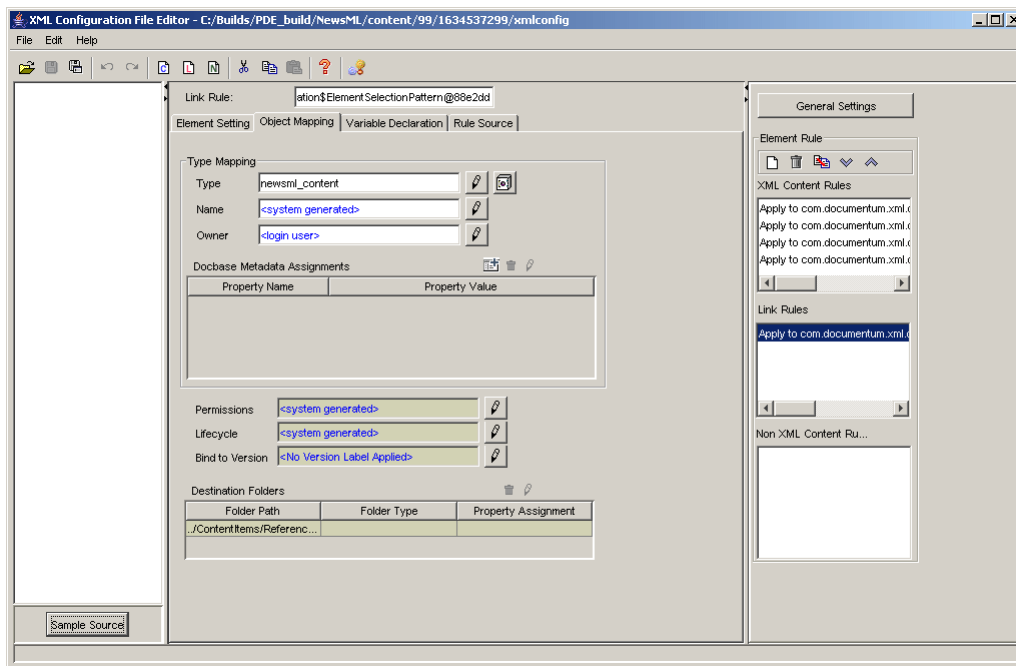
An element rule or entity rule instructs the XML application to preserve all external parsed (XML) entities as separate components of the XML virtual document when imported or checked in, and to maintain their status as external entities when the main XML document is exported or checked out.

For more information about XML applications and XML application configuration files, see the *Documentum XML Applications Development Guide*.

### To view or modify an XML application configuration file:

1. In your project, expand the **XML Applications** folder.
2. Open the **.xmlapplications** file you want to modify in one of the following ways:
  - Double-click the **.xmlapplications** file.
  - Right-click the **.xmlapplications** file and select **Open** from the menu.

The **XML Configuration File** editor appears.



3. Modify the XML configuration file as desired, then save your changes.

**Note:** If you added or deleted references to repository objects in the XML application configuration file, make the same modifications in your project.



# Building and Installing a Project

This chapter contains the following topics:

- [Understanding the build and installation process, page 199](#)
- [Configuring the project installation options, page 200](#)
- [Configuring artifact install options, page 203](#)
- [Generating a DAR file, page 205](#)
- [Installing a project, page 205](#)
- [Creating an installation parameter file, page 208](#)

## Understanding the build and installation process

Composer projects must be built and installed in a repository in order to run. During the build process, Composer generates an executable version of the project. This executable version of a Composer project is called a Documentum Archive (DAR) and consists of one single file that contains the executable binary files of the project. There are two ways to install a Composer project:

- Composer user interface

The Composer user interface lets you install a Documentum project in a repository. As part of the installation process, you configure certain installation parameters that apply to the whole project and to individual artifacts, as described in [Configuring the project installation options, page 200](#) and [Configuring artifact install options, page 203](#).

You also have the option to build the application and generate a DAR archive file (.dar), as described in [Generating a DAR file, page 205](#) for use with the DAR Installer or headless Composer.

- DAR Installer

The DAR Installer cannot build projects, but can install pre-built DAR files that were built with Composer or headless Composer.

- Ant tasks and headless Composer

Ant tasks and headless Composer let you build a project, generate a DAR file, and install the DAR file into a repository using a command-line interface that is useful for automation. For more

information about using Ant tasks and DAR files, see [Chapter 21, Managing Projects and DAR Files Using Ant tasks and Headless Composer](#).

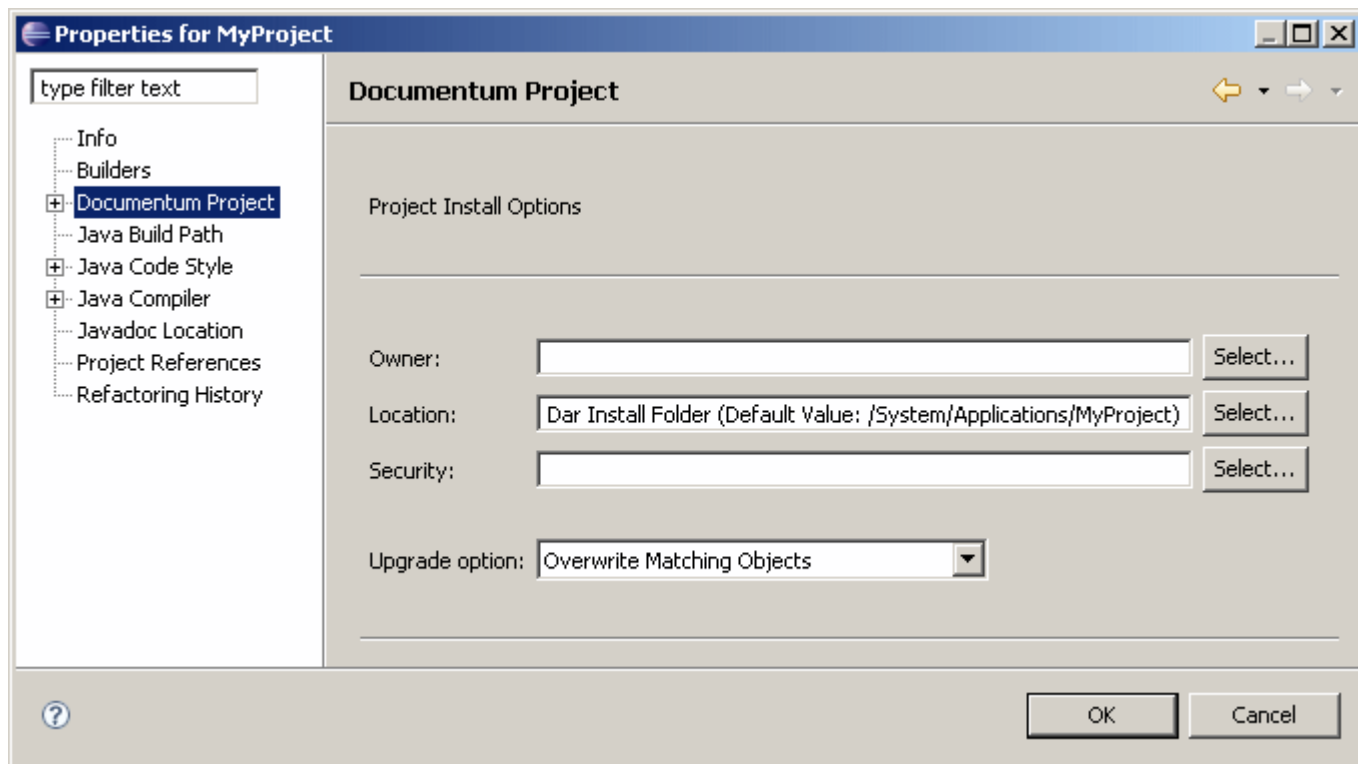
## Configuring the project installation options

The project installation options let you set installation parameters that apply to the entire project, such as DFS module options, pre- and post-installation procedures and upgrade options.

### To configure project installation options:

1. Open the project **Properties** dialog using one of two options, as follows:
  - In the Composer main menu, select **Project > Properties**.
  - Right-click the name of the project you want to install and select **Properties** from the drop-down menu.

The **Properties** dialog appears.



2. Specify the installation options for the project, as described in [Table 60, page 201](#).



**Table 60. Project installation options**

Install Option	Description
Owner	Specifies the owner installation parameter for installing this project. The owner must be a valid user in the repository where the project is installed. Click <b>Select ...</b> to select an owner from the listbox. For more information about adding a new owner installation parameter, see <a href="#">Adding an owner installation parameter, page 201</a> .
Location	Specifies the location installation parameter for installing this project. Click <b>Select ...</b> to select a location from the listbox or accept the default value.
Security	Specifies the permission set (ACL) parameter for installing this project. Click <b>Select ...</b> to select a location from the listbox.
Upgrade option	Specifies the upgrade option used when installing this project. There are three upgrade options, as follows: <ul style="list-style-type: none"> <li>• <b>Overwrite Matching Objects</b> — Overwrites all objects in the repository have matching objects in the project. If there are new objects in the project, they are installed as new objects in the repository.</li> <li>• <b>Ignore Matching Objects</b> — Ignores all objects in the repository that have a matching object in the project. If there are new objects in the project, they are installed as new objects in the repository.</li> <li>• <b>Create New Version Of Matching Objects</b> — Creates a new version of all objects in the project that have a matching object in the repository. If there are new objects in the project, they are installed as new objects in the repository.</li> </ul>

3. Double-click **Documentum Project** to expand the menu tree.
4. Select **Install Procedure** to specify a pre- and a post-installation procedure, if required, and enter the procedure names in the associated Pre-installation procedure and Post-installation procedure fields.
5. Click **OK** to save your project installation options.

## Adding an owner installation parameter

You can add an owner installation parameter using the **Owner Installation Parameter Artifact** dialog.

### To add an owner installation parameter:

1. Click **Select ...** next to the **Owner** field in the Project Install Options wizard.  
The **Owner Installation Parameter Artifact** dialog appears.
2. Click **New**.  
The **New Documentum Artifact – Name and Location** dialog appears.

- Accept the default folder and artifact name by clicking **Next**.  
The **Installation Parameter Artifact** dialog appears.

- Enter the parameter name, type, optional description, and default value, as follows:

**Table 61. Owner installation parameters**

Parameter	Description
Parameter name	A string specifying the name of the owner installation parameter.
Parameter type	A string specifying the type of the owner installation parameter. The type is set to <b>Owner</b> by default and cannot be changed.
Description	An optional description of the owner installation parameter.
Default value	An optional default value for the owner installation parameter. If you specify a default value for the owner installation parameter, the owner must be a valid user in the repository where the project is installed.

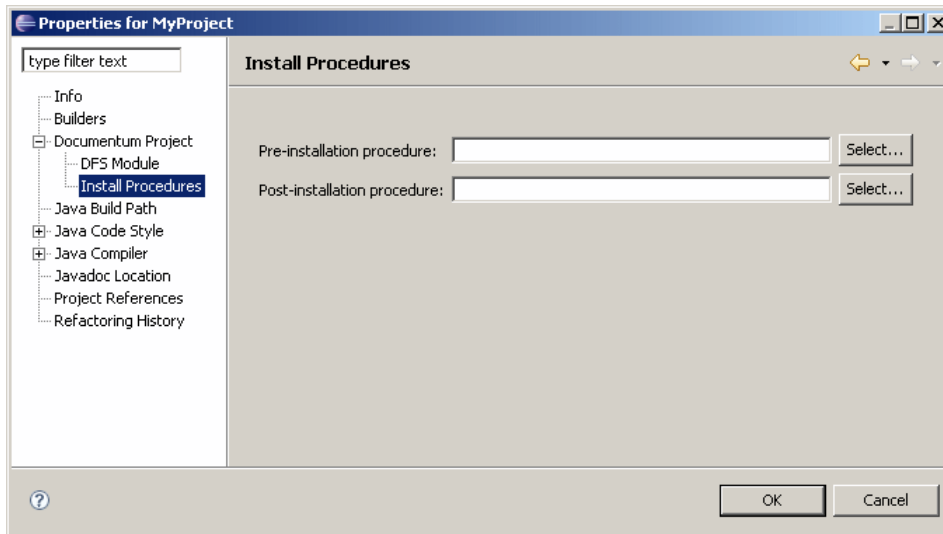
- Click **Finish**.

## Configuring pre- and post-installation procedures

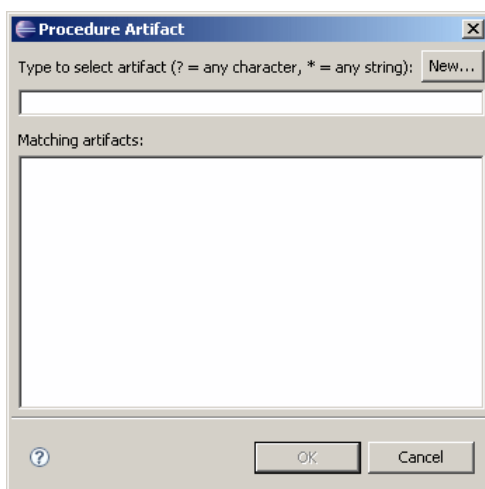
You can configure pre- and post-installation procedures for a project in the **Install Procedures** dialog.

### To configure pre- and post-installation procedures:

- Right-click the project and select **Properties** from the drop-down list.  
The **Properties** dialog appears ([Figure 2, page 34](#)).
- Expand **Documentum Project** and select **Install Procedures**.  
The **Install Procedures** dialog appears.



3. Click the **Select** button next to the **Pre-installation procedure** or **Post-installation procedure** field. The **Procedure Artifact** dialog appears.



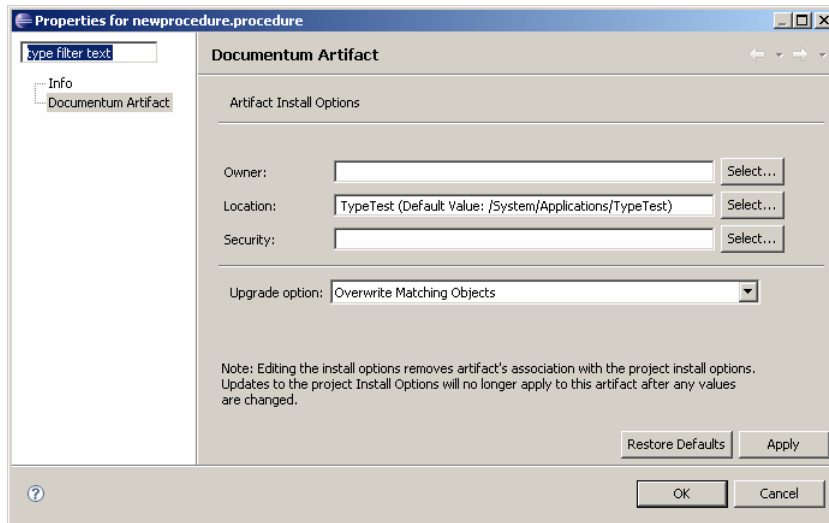
4. Select a procedure from the **Matching Artifacts** list or click **New** to create a procedure.
5. Click **OK**.

## Configuring artifact install options

Composer lets you configure installation options not only at a project level, but also for each individual artifact. The installation option for an artifact overrides the project installation option.

### To configure install options for an individual artifact:

1. In the **Documentum Navigator** view, right-click on the artifact for which you want to configure installation options, then select **Properties** from the drop-down list.  
The **Documentum Artifact** properties dialog appears.



2. Configure the install options for the artifact, as described in [Table 62, page 204](#).

**Table 62. Artifact installation options**

Install Option	Description
Owner	Specifies the owner installation parameter for installing this artifact. The owner must be a valid user in the repository where the artifact is installed. Click <b>Select ...</b> to select an owner from the listbox. For more information about adding a new owner installation parameter, see <a href="#">Adding an owner installation parameter, page 201</a> .
Location	Specifies the location installation parameter for installing this artifact. Click <b>Select ...</b> to select a location from the listbox.
Security	Specifies the permission set (ACL) parameter for installing this artifact. Click <b>Select ...</b> to select a location from the listbox.
Upgrade option	<p>Specifies the upgrade option used when installing this artifact. There are three upgrade options, as follows:</p> <ul style="list-style-type: none"> <li>• <b>Overwrite Matching Objects</b>—Overwrites all objects in the repository have matching objects in the project. If there are new objects in the project, they are installed as new objects in the repository.</li> <li>• <b>Ignore Matching Objects</b>—Ignores all objects in the repository that have a matching object in the project. If there are new objects in the project, they are installed as new objects in the repository.</li> <li>• <b>Create New Version Of Matching Objects</b>—Creates a version of all objects in the project that have a matching object in the repository. If there are new objects in the project, they are installed as new objects in the repository.</li> </ul> <p><b>Note:</b> If you are configuring the installation options for a smart container artifact, be sure to set the upgrade option to <b>Create New Version Of Matching Objects</b>. Do not select <b>Overwrite Matching Objects</b> for smart container artifacts because overwriting smart</p>

Install Option	Description
	container objects invalidates the model-instance association for existing instances.

- Click **OK** to save your changes.

## Generating a DAR file

A DAR file is the executable version of a project that gets installed in a Documentum repository. A DAR file contains only the binary files of a project but not the source files, so you cannot convert a DAR file into a Composer project. You can install a DAR file with the DAR Installer or headless Composer.

If you have the **Project > Build Automatically** option turned on, you can obtain the **<project>.dar** file from the **...\<workspace>\<project>\bin-dar** directory of the Composer workspace. It is recommended that you leave this on in most situations. If you have the **Project > Build Automatically** option turned off, complete the following steps:

### To generate a DAR file:

- Right-click the project you want to build.
- Select **Build Project** from the drop-down list.  
Composer builds the project and generates a **<project>.dar** file in the **...\<workspace>\<project>\bin-dar** directory, where **<workspace>** is the name of your workspace and **<project>** is the name of your project.

## Installing a project

After you have created your project and you are ready to deploy the application, build and install the application in a repository. If you are using a source control system, check out the project from source control before you build and deploy the application.

**Note:** If you have projects that reference each other, be sure to install the projects in the correct order. For example, if project B references artifacts in project A, then project A must be installed first.

### To install a project:

- In the **Documentum Project Navigator** view, right-click the project you want to install and select **Install Documentum Project ...** from the drop-down menu. Composer automatically builds the project in the background. Any errors during the build process are displayed in the **Error** view. The **Installation Settings** dialog appears.

**Install Wizard**

**Installation Settings**  
Enter required fields and click Next to edit the Installation Parameter values for this repository

**Repository Details**  
Enter repository information, installation parameter file, and installation options

Select or enter the repository:

Repository name:

User name:

Password:

Domain:

Login

**Installation option**

**Installation Parameter File Details**

☐ Use an Installation Parameter File

Installation Parameter File:  Browse...

**Localization**

☐ Install Localized Artifact Data

Locale Properties Folder:  Browse...

< Back Next > Finish Cancel

- Enter the installation information, as described in [Table 63, page 206](#).

**Table 63. Install parameter information**

Install Parameter	Description
Repository	The name of the installation repository. Mandatory parameter. Type the repository name or select a repository from the drop-down list. You must have SUPERUSER privileges to access the repository.
User name	The login user name for the repository.
Password	The login password for the repository.
Domain	The domain name of the repository. If the repository resides in a different domain than the client from which the repository is accessed, specify the domain name.

Install Parameter	Description
Install options	<p>Specifies how the project is installed in the repository. There are three install options, as follows:</p> <ul style="list-style-type: none"> <li>• <b>Use Project and Artifact Settings</b>—Installs the project according to the options that are configured using the Project Install Option dialog in the project properties.</li> <li>• <b>Overwrite</b>—If the project exists in the repository, all objects are overwritten when a modified version of the project is installed.</li> <li>• <b>Version</b>—If the project exists in the repository, objects that are versionable are versioned when a modified version of the project is installed. Objects that are not versionable are overwritten.</li> </ul>
Use an Installation Parameter File	<p>Select this option if you want to use an installation parameter file. Optional parameter. Click <b>Browse</b> and select the installation parameter file. If you are installing this project for the first time, and you want to use an input parameter file, create the input parameter file first, as described in <a href="#">Creating an installation parameter file, page 208</a>.</p>
Install Localized Artifact Data	<p>Select this option if you want to localize your project. Optional parameter. Click <b>Browse</b> and select the Locale Properties Folder containing the localized files. For more information about localizing a project, see <a href="#">Localizing a Composer project, page 34</a>.</p>

3. Click **Next**.

The **Edit Installation Parameter File Values** dialog appears.

**Install Wizard**

**Edit Installation Parameter File Values**

Update Parameters by typing new value within the Override column. Once Override values are entered, you must save these values as an Installation

Installation Parameter File: None Selected

Parameter Name	Parameter Type	Description	Default Value	Override Value
TestRelationTy...	Folder	default l...	/System/A...	
filestore_01	Storage		filestore_01	

Save Override values to Installation Parameter File:

Note: Pressing Prev button will clear all the overrided values entered.

< Back   Next >   Finish   Cancel

The installation parameter table lists the name, type, and default value for each installation parameter.

**Note:** You can change the default value for each installation parameter and save the new value to an input parameter file. For more information about creating an input parameter file, see [Creating an installation parameter file, page 208](#).

4. Click **Finish** to install the project in the repository.

**Note:** If you install a project in the repository, then edit the project in Composer and remove one or more objects and re-install the project, the old objects remain in the repository. Composer does not delete objects from the repository during installation, even if the objects were removed from the project itself.

## Creating an installation parameter file

If you want to change the default installation values for one or more installation parameters, create an installation parameter file. The installation parameter file contains the name, type, and default value for each installation parameter in the project.

### To create an installation parameter file:

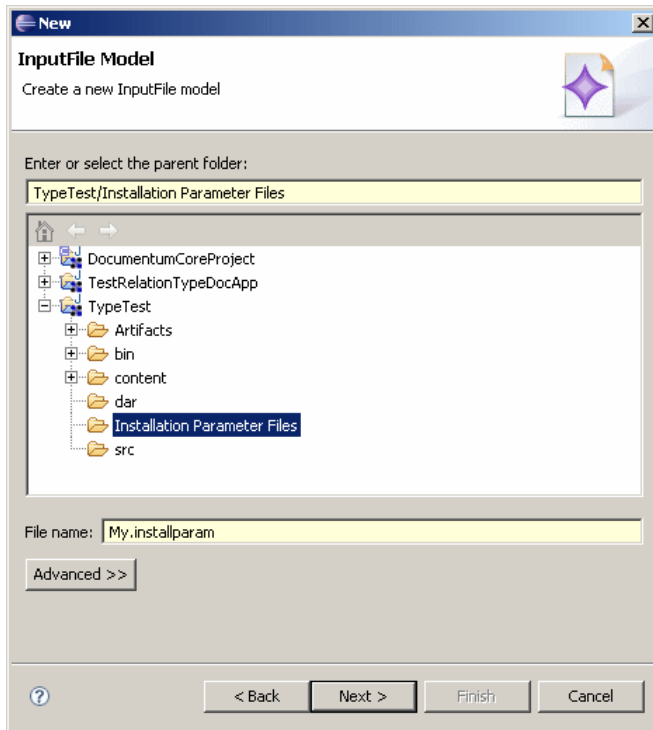
1. Open the **Installation Parameter File** wizard in one of the following ways:
  - From the Composer menu, select **File > New > Other**.
  - In your Composer project, right-click **Installation Parameter Files**. Select **New > Other** from the drop-down menu.

The **Select a wizard** dialog appears.

2. Double-click **Installation Parameter File** to expand it, select **Installation Parameter File New Wizard**, then click **Next**.

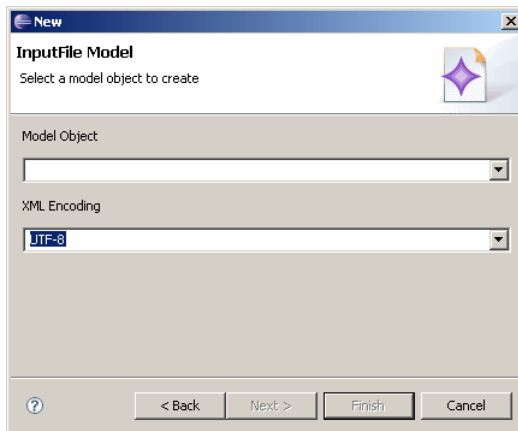
The **Input File** dialog appears.





3. Select a parent folder for your installation parameter file and enter a file name in the **File name:** field, then click **Next**.

The **Input File Model** dialog appears.



4. Select **Installation Parameter File** from the **Model Object** drop-down list, and **UTF-8** for XML encoding, then click **Finish**.

Composer creates an input parameter file and displays the file in the **Resource Set** view.

## Installing a DAR file with the DAR Installer

A DAR file is a deployable, package representation of a Composer project. You can use the DAR Installer to install a DAR file to a repository if you do not want to use the interface within Composer.

The DAR Installer is useful for installing Documentum product DAR files or in cases where you want to decouple the development of DAR files from the installation of DAR files. The DAR Installer requires Composer to be installed, but does not launch the full Composer IDE.

You can also install a DAR file with headless Composer. For more information on installing DAR files with headless Composer, see [Chapter 21, Managing Projects and DAR Files Using Ant tasks and Headless Composer](#).

When you open the DAR Installer, it creates three folders in your Composer installation directory:

- `darinstallerconfig` – contains configuration files for the DAR Installer Plugin
- `darinstallerlogs` – the default location of the log files
- `darinstallerworkspaces` – workspaces that are created and used by the DAR Installer Plugin. The DAR Installer Plugin does not delete these workspaces automatically after installation of the DAR file. The workspace directories are named in the following form: `darinstaller_workspace_YYYY-MM-DD-HH-MM-SS`. Moving, deleting, or adding projects manually to the workspace might have adverse effects on DAR installations.

**Note:** If the DAR file that you are installing depends on other, required DAR files, install those required DAR files first. If DAR files that you want to install depend on DAR files that you have already installed, place the reference DAR files in the same location as the DAR file that you want to install. The DAR installer looks for them there.

The DAR Installer requires you to fill in certain values that are marked with an asterisk (\*). All other fields are optional. For a description of the fields for the DAR Installer plug-in, see [Table 64, page 210](#).

### To install a DAR file:

1. Run `darinstaller.exe`, which is located in the Composer root directory, to start the DAR Installer.
2. In the **DAR Details** section, specify values for the fields.
3. In the **Connection Broker Details** section, specify values for **Connection Broker Host** and **Connection Broker Port** and click **Connect**.
4. In the **Repository Details** section, specify values for the fields and click **Install** to install the DAR file to the repository.

You can view the log for the DAR installation by selecting the log file from the **Log File** drop down menu and clicking **Open**.

**Table 64. DAR Installer fields**

Parameter	Required	Description
DAR	Yes	The absolute file path to the .dar file that you want to install. The file path cannot contain any I18N characters or the installation will fail.
Input File	No	The absolute file path to the install-based parameter file.

Parameter	Required	Description
Local Folder	No	The absolute file path to localized .properties files. If you want to make your application available in other languages, localize the project data such as labels, tabs, and descriptions.
Log File	No	The file to save the log to. If this is not specified, the file defaults to <DAR>.log.
Connection Broker Host	Yes	The address of the Connection Broker.
Connection Broker Port	Yes	The port of the Connection Broker Repository.
Repository	Yes	The name of the repository that you want to install the DAR file to. Click the Connect button after entering the Docbroker host and port to retrieve the available repositories.
User Name	Yes	The login name for the repository.
Password	Yes	The password for logging in to the repository.
Domain	No	The domain of the user.



# Managing Projects and DAR Files Using Ant tasks and Headless Composer

Headless Composer is the non-UI command line version of Composer that includes a set of Ant tasks for common build and deployment features of Composer, such as import, build, and install.

## Creating a headless Composer build

A headless Composer build allows you to automate the build and installation of Composer projects. A headless Composer build consists mainly of two parts: Ant scripts that define the build and a batch file that sets up the build environment and runs the Ant scripts.

## Creating Ant scripts to build, modify, and install Composer projects

Ant scripts are XML files that define your build. Composer provides Ant tasks that allow you to call certain Composer functionality from an automated build.

In general, create two separate Ant build files that build your projects and install your projects. The Ant scripts should be encoded in UTF-8 to ensure proper functionality.

### To create the Ant scripts:

1. Create a file named build.xml.
2. Create a target to import the projects that you want to work with into the headless Composer workspace with the `emc.importProject` task. You can also create new projects with the `emc.createArtifactProject` task. You either have to create or import a project into the workspace before you call any other Ant task.
3. Make modifications that you want to the project with the `emc.importArtifacts` or `emc.importContent` tasks.
4. Create a target to build the project with the `emc.build` task. Call this task before the `emc.dar` task to ensure that the DAR file contains the latest built code.

5. Create a target to generate the DAR file with the emc.dar task. Call this task before the emc.install task to ensure that the code built from emc.build task makes it into a new DAR file that you can install later.
6. Create a file named install.xml.
7. Create a target to install the DAR file with the emc.install task.

**Note:** The Ant script in the following example has a default target specified so when you call the script, it runs the target automatically. This default target is named “package-project” and depends on another target, “build-project.” This means that the “build-project” target is always ran before “package-project”. The “build-project” target in turn depends on “update-jardef,” which depends on “import-project”. These targets are chained together so that the “import-project” target always runs first. The order in which the Composer Ant tasks are run are important, so adhere to the general order described in the previous procedure.

#### Example 21-1. Example build.xml file

```
<?xml version="1.0"?>
<project name="HelloWorldBuild" default="package-project">

  <target name="import-project" description="
    Must import a project before updating, building, or installing it">
    <emc.importProject dmpproject="HelloWorldArtifacts" failonerror="true"/>
  </target>

  <target name="update-jardef" depends="import-project" description="
    Update JARDef with most current JAR file">
    <emc.importContent dmpproject="HelloWorldArtifacts"
      artifactpath="Artifacts/JAR Definitions/hello.jardef"
      contentfile="build_workspace/HelloWorldBOFModule/bin-impl/hello.jar" />
  </target>

  <target name="build-project" depends="update-jardef"
    description="Build the project">
    <emc.build dmpproject="HelloWorldArtifacts" failonerror="true"/>
  </target>

  <target name="package-project" depends="build-project"
    description="Package the project into a DAR for installation">
    <delete file="HelloWorld.dar" />
    <emc.dar
      dmpproject="HelloWorldArtifacts"
      manifest="bin/dar/default.dardef.artifact"
      dar="HelloWorld.dar" />
  </target>
</project>
```

#### Example 21-2. Example install.xml file

```
<?xml version="1.0"?>
<project name="headless-install" default="install-project">
  <target name="install-project"
    description="Install the project to the specified repository.
    dfc.properties must be configured">
    <emc.install
      dar="HelloWorld.dar"
      docbase="GlobalRegistry"
      username="Administrator"
      password="emc"
      domain="" />
  </target>
```

```
</project>
```

**Note:** If you are installing a DAR file that depends on other reference DAR files, install the reference DAR files first and in the same Ant script as the DAR file that you want to install. You must do this even if you previously installed the reference DAR files.

## Creating a batch file to setup and run the build

The batch file is used to configure the environment variables and workspaces on your local machine, and calls the Ant scripts that contain the import, build, or install instructions for your build.

The following example batch file performs these actions:

- Defines the Eclipse directory path to be at C:\ComposerHeadless.
- Creates two workspaces, one for importing and building a project (BUILDWORKSPACE) and one for installing the resulting DAR file (INSTALLWORKSPACE).
- Specifies the location of the build and installation scripts: build.xml and install.xml.
- Cleans the workspaces.
- Copies the Composer projects into the build workspace.
- Runs the build and installation scripts.

### Example 21-3. Example batch file

```
REM Set environment variables to apply to this command prompt only
SETLOCAL

REM Sets the root location of headless Composer
SET ECLIPSE="C:\ComposerHeadless"

REM Sets the location of your source projects.
REM This location gets copied into your build workspace directory
SET PROJECTSDIR="C:\Documents and Settings\Administrator\composer-workspace"

REM Sets the workspace directory where Composer builds the projects
REM that you want to install to a repository
SET BUILDWORKSPACE="C:\ComposerHeadless\example build\build_workspace"

REM Sets the workspace directory where Composer extracts built DAR files
REM before installing them to a repository
SET INSTALLWORKSPACE="C:\ComposerHeadless\example build\install_workspace"

REM Sets the Ant script that builds your projects
SET BUILDFILE="C:\ComposerHeadless\example build\build.xml"

REM Sets the Ant script that installs your projects
set INSTALLFILE="C:\ComposerHeadless\example build\install.xml"

REM Delete old build and installation workspaces
RMDIR /S /Q %BUILDWORKSPACE%
RMDIR /S /Q %INSTALLWORKSPACE%

REM Copy source projects into build workspace
XCOPY %PROJECTSDIR% %BUILDWORKSPACE% /E

REM Run Ant scripts to build and install the projects
JAVA -cp %ECLIPSE%\startup.jar org.eclipse.core.launcher.Main -data
```

```
%BUILDWORKSPACE% -application org.eclipse.ant.core.antRunner -buildfile %BUILDFILE%  
JAVA -cp %ECLIPSE%\startup.jar org.eclipse.core.launcher.Main -data  
%INSTALLWORKSPACE% -application org.eclipse.ant.core.antRunner -buildfile %INSTALLFILE%
```

## emc.importProject

The `emc.importProject` task imports a Documentum project into the headless Composer workspace. A project must be imported into the workspace before you can build or install it.

### Example 21-4. Syntax

Attributes in brackets are optional.

```
<emc.importProject  
  dmpproject="project_name"  
  [failonerror="true|false"]/>
```

**Table 65. `emc.importProject` task**

Parameter	Description
<code>dmpproject</code>	Specify the name of the project that you want to import.
<code>failonerror</code> (optional)	Specify true if you want the build to fail if the project does not exist, false otherwise. The default behavior is false.

## emc.createArtifactProject

The `emc.createArtifactProject` task creates a Composer project.

### Example 21-5. Syntax

Elements and attributes in brackets are optional.

```
<emc.createArtifactProject name="project_name" [overwrite="true|false"]>  
  [<projectReferences [failOnMissingReferences="true|false"]>  
    <project name="reference_project_name"/>  
  </projectReferences>  
</emc.createArtifactProject>
```

The `emc.createArtifactProject` task specifies the project to create. It can also contain the `projectReferences` element that specifies reference projects for the project that you are creating.

**Table 66. `emc.createArtifactProject` task**

Parameter	Description
<code>name</code>	Specify the name of the project that you want to create.



Parameter	Description
overwrite	Specify <b>true</b> if you want existing projects overwritten, <b>false</b> otherwise. The default behavior is set to <b>false</b> .
projectReferences element (optional)	Specify this element if you want to assign reference projects to the project. Reference projects must be imported into the workspace using the <code>emc.importProject</code> command.

The `projectReferences` element contains project elements that specify reference projects for the project that you are creating. A `projectReferences` element can contain multiple project elements.

**Table 67. `projectReferences` element**

Parameter	Description
failOnMissingReferences (optional)	Specify <b>true</b> if you want the build to fail if the reference projects are not available, <b>false</b> otherwise. The default behavior is set to <b>false</b> .
project element	Specify a project element for each project that you want to designate as a reference project.

The project element specifies the name of the project that you want to designate as a reference project. You can specify multiple project elements.

**Table 68. `project` element**

Parameter	Description
name	Specify the name of the project that you want to designate as a reference project. The project must already be imported into the headless Composer build workspace with the <code>emc.importProject</code> task.

#### Example 21-6. Example

The following target creates a project named “Test.” It sets the `overwrite` attribute to `true`, which overwrites any existing project with that name. It defines “MyReferenceProject” as a reference project and sets the `failOnMissingReferences` attribute to `true`, which causes the creation of the project to fail if the reference project does not exist.

```
<emc.createArtifactProject name="Test" overwrite="true">
  <projectReferences failOnMissingReferences="true">
    <project name="MyReferenceProject"/>
  </projectReferences>
</emc.createArtifactProject>
```

## emc.createTaskSpaceApplicationProject

The `emc.createTaskSpaceApplicationProject` task creates a Composer project from a TaskSpace application.

#### Example 21-7. Syntax

Elements and attributes in brackets are optional.

```
<emc.createTaskSpaceApplicationProject name="project_name" docbase="repo_name"
```

```

username="username" password="password">
  [<projectReferences>
    <project name="reference_project_name"/>
  </projectReferences>]
</emc.createTaskSpaceApplicationProject>

```

The `emc.createTaskSpaceApplicationProject` task specifies a project to create from an existing TaskSpace application in a repository. It gives the Composer project the same name as the TaskSpace application. You can also declare the `projectReferences` element to specify reference projects for the project that you are creating. Reference projects must be imported into the workspace before you can use them.

**Table 69. `emc.createArtifactProject` task**

Parameter	Description
name	Specify the name of the project that you want to create.
docbase	The name of the repository where the TaskSpace application resides.
username	The username to login to the repository.
password	The password for the username.
projectReferences element (optional)	Specify this element if you want to assign reference projects to the project. Reference projects must be imported into the workspace using the <code>emc.importProject</code> command.

The `projectReferences` element contains project elements that specify reference projects for the project that you are creating. A `projectReferences` element can contain multiple project elements.

**Table 70. `projectReferences` element**

Parameter	Description
project element	Specify a project element for each project that you want to designate as a reference project.

The project element specifies the name of the project that you want to designate as a reference project. You can specify multiple project elements.

**Table 71. `project` element**

Parameter	Description
name	Specify the name of the project that you want to designate as a reference project. The project must already be imported into the headless Composer build workspace with the <code>emc.importProject</code> task.

#### Example 21-8. Example

```

<emc.createTaskSpaceApplicationProject name="TaskSpaceApp" docbase="repository"
username="dmadmin" password="n0lif3">
  <projectReferences>
    <project name="TSWorkflows"/>
  </projectReferences>
</emc.createTaskSpaceApplicationProject>

```

## emc.importArtifacts

The emc.importArtifacts task imports artifacts from a repository into a specified project.

### Example 21-9. Syntax

Elements and attributes in brackets are optional.

```
<emc.importArtifacts project="project_name" docbase="repo_name"
username="username" password="password" [domain="xyz"] >
  <objectIdentities>
    [<id value="object_id"/>]
    [<path value="object_path"/>]
    [<qualification value="dql_qualifier"/>]
  </objectIdentities>
</emc.importArtifacts>
```

**Table 72. emc.importArtifacts task**

Parameter	Description
project	Specify the name of the project that you want to import the artifact into.
docbase	Specify the repository name where the artifact resides.
username	Specify the user to login to the repository with.
password	Specify the password to login to the repository with.
domain	Specify the domain of the user.
objectIdentities element	Contains id, path, or qualification elements that specify the identities of the artifacts to import.

The objectIdentities element contains the identities of the object in the repository that you want to import. Each identity element has one attribute, value, that specifies the value of the identity.

**Table 73. objectIdentities element**

Parameter	Description
id element	Used to specify the object ID of the artifact in the repository.
path element	Used to specify the path of the artifact in the repository.
qualification element	Used to specify the DQL qualifier for the artifact in the repository.

### Example 21-10. Example

For example, the following target imports artifacts from test\_repository into the Test project. The artifacts to import are declared with the objectIdentities, which specifies one object by object ID, one by path (i.e. must be a sysobject), and one by DQL qualifier (typically used for importing non-sysobjects such as a type or group).

```
<emc.importArtifacts project="Test" docbase="test_repository"
username="dmadmin" password="n0lif3">
  <objectIdentities>
    <id value="08000001800737f7"/>
    <path value="/System/Applications/Collaboration Services/CreateAcl"/>
    <qualification value="dm_group where group_name = 'my_group'"/>
  </objectIdentities>
</emc.importArtifacts>
```

## emc.importContent

The emc.importContent task imports or updates a content file in a specified artifact and project. Currently, the emc.importContent command only supports importing content into procedure artifacts and JAR definitions.

### Example 21-11. Syntax

```
<emc.importContent
  dmproject="project_name"
  artifactpath="path/to/artifact"
  contentfile="path/to/content"
  contentid="content_id" />
```

**Table 74. emc.importContent task**

Parameter	Description
dmproject	Specify the name of the project containing the artifact into which the content is imported.
artifactpath	Specify the path name of the artifact relative to the project, including the name of the artifact itself, for example: Artifacts/JAR Definitions/myjardef.
contentfile	Specify the absolute file path of the content file to import. The file must exist in a readable format. If specifying a relative path, the path is relative to the location of the Ant script.
contentid (optional)	Specify the identifier of the content. If specified, the contentid is used to uniquely name the content within the content store of the project. If no contentid is specified, Composer generates and assigns a random name.

## emc.build

The emc.build task builds a Composer project, which can then be passed into the emc.dar task to generate a DAR file.

### Example 21-12. Syntax

Elements and attributes in brackets are optional.

```
<emc.build
  dmproject="project_name"
  [failonerror="true|false"] />
```

**Table 75. emc.build task**

Parameter	Description
dmproject	Specify the name of the project that you want to build.
failonerror (optional)	Specify true if you want the build to fail if the project does not build correctly, false otherwise.

## emc.dar

The emc.dar task generates a DAR file from a Documentum project. The emc.dar task depends on the output from the emc.build task and must run in the same Java process. Define both tasks in the same Ant script and call them within the same build to ensure proper functionality.

### Example 21-13. Syntax

```
<emc.dar
  dmproject="project_name"
  manifest="bin/dar/project_name.dardef.artifact|default.dardef.artifact"
  dar="path/to/dar/dar_name.dar" />
```

**Table 76. emc.dar task**

Parameter	Description
dmproject	Specify the name of the project that you want to build a DAR file from.
manifest	Specify the relative file path to the <i>project.dardef.artifact</i> or <i>default.dardef.artifact</i> file that is located in the bin/dar directory of your project. This file is usually named <i>default.dardef.artifact</i> unless the project was created by importing a DocApp from a repository. In this case, the file is named <i>project.dardef.artifact</i> , where <i>project</i> is the name of your project. However, an empty <i>default.dardef.artifact</i> file is still created in this case.
dar	The absolute file path to the target .dar file. The .dar file must not exist yet.

## emc.install

The emc.install task installs a project's DAR file into a repository.

### Example 21-14. Syntax

Attributes in brackets are optional.

```
<emc.install
  dar="path/to/dar/dar_name.dar"
  docbase="repository"
  username="user"
  password="password"
  [domain="domain"] />
[inputfile="path/to/dar/filename.installparam"]
[localesFolder="/path/to/locales/folder"] />
```

**Note:** If you are installing a DAR file that depends on other reference DAR files, install the reference DAR files first and in the same Ant script as the DAR file that you want to install. You must do this even if you previously installed the reference DAR files.

**Table 77. emc.install task**

Parameter	Description
dar	The absolute file path to the .dar file being installed. The file path must contain only Unicode (UTF-8) characters or the installation fails.
docbase	The name of the repository into which the .dar file is installed.
username	The login name for the repository.
password	The password for logging in to the repository.
localesFolder (optional)	The absolute file path to localized .properties files. If you want to make your application available in other languages, localize the project data, for example labels, tabs, and descriptions. For more information about localizing a project, see <a href="#">Localizing a Composer project, page 34</a> .
inputfile (optional)	The absolute file path to the install-based parameter file.
domain (optional)	The domain in which the repository resides.

If you want to enable debugging messages during the installation, call the following Ant task to set logging preferences:

```
<emc.preferences logTraceMessages="false" logDebugMessages="true" />
```

## Installing a DAR file with headless Composer on UNIX and Linux systems

The headless Composer distribution that is bundled with Content Server for UNIX or Linux can be used to install a DAR file to UNIX, Linux, or Windows systems. Scripts are provided for setting up the environment and installing the DAR file if you do not want to go through the trouble of creating your own deployment scripts. To install a DAR file with headless Composer on UNIX and Linux systems:

1. Login to the Content Server system as the owner of the repository that you want to install the DAR file to.
2. Ensure that your environment variables are set according to the Content Server deployment guide. Most notably, run the \$DM\_HOME/bin/dm\_set\_server\_env.sh shell script to set the environment variables
3. Run the following command to install a DAR file, replacing the variables with the appropriate values for your environment:

```
$JAVA_HOME/bin/java -Ddar=$PATH_TO_DAR_FILE -dlogpath=$PATH_TO_LOG_FILE
-Ddocbase=$REPOSITORY_NAME -Duser=$REPOSITORY_SUPER_USER
-Ddomain=$REPOSITORY_USER_DOMAIN -Dpassword=$PLAIN_TEXT_PASSWORD
-cp $DM_HOME/install/composer/ComposerHeadless/startup.jar
org.eclipse.core.launcher.Main -data $DM_HOME/install/composer/
```

```
workspace -application org.eclipse.ant.core.antRunner -buildfile  
$DM_HOME/install/composer/deploy.xml
```

**Note:** If you are installing a DAR file that depends on other reference DAR files, you cannot use the supplied deploy.xml script. Create your own Ant script that contains targets to install the reference DAR files first and then installs the DAR file. These targets must be in the same Ant script and must be run in the same call to Ant.





# Working with Source Control Systems

This chapter contains the following topics:

- [Using a source control system, page 225](#)
- [Building the project, page 227](#)

## Using a source control system

If a team of developers is working on the same Composer project and other referenced projects, they often use a source control system to enable collaboration. Most source control systems offer Eclipse-based plug-ins that support an integrated development environment. These plug-ins can be used with Documentum Composer, as long as the plug-ins are compatible with Eclipse 3.2.

## Checking in projects

The following guidelines are recommended when using a source control system:

- Check in the project, not the Composer workspace since workspaces are personalized
- Check in the following Documentum project directories and files:
  - Artifacts
  - content
  - dar
  - Web Services
  - src (even if this folder is empty)
  - all non-directory files at the root of the project (such as .classpath, .dmproject, .project, and .template)

Other directories, such as bin and bin-dar, and the files that are in them are derived resources and should not be checked in.

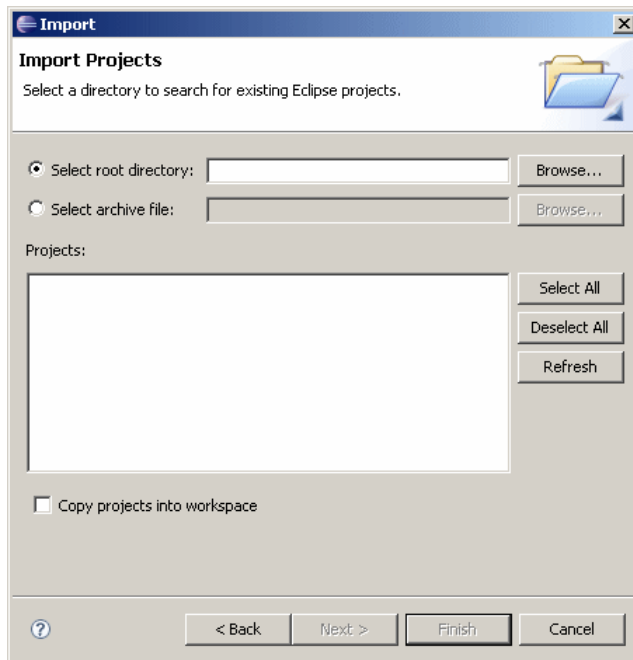
## Checking out and importing projects

When working with a source control system, import the projects into a Composer workspace after retrieving them from the source control system.

### To check out and import projects into a Composer workspace:

1. Check out the project and any referenced projects from the source control system.
2. In Composer, select **File > Import > Documentum > Existing projects into workspace**, then click **Next**.

The **Import Projects** dialog appears.



3. Select the **Select root directory** field and enter the path to the root directory where your checked out projects are located, or click **Browse** to search for the root directory. The available projects display in the **Projects** list box.

**Note:** If you do not see the projects that you want to import in the **Projects** list box, verify that you do not have a previous version of the project in your Composer workspace. You can only import projects that do not exist in your local Composer workspace.

4. Select the projects you want to import then click **Finish**.

Do not check the **Copy projects into workspace** option.

# Building the project

Building a project is analogous to compiling code. During the build process Composer validates the artifacts and report any validation errors. The build process can be initiated in one of two ways:

- From the Composer UI
- Using headless Composer and Ant tasks

Using ANT tasks to build the DAR file is the more efficient option if the Composer projects are part of a significant build/test/deploy development cycle and the projects are on a nightly build schedule. For more information about building a DAR file using Ant tasks, see [emc.build, page 220](#).



# Glossary

---

**ACL**

An access control list (ACL) specifies the access each user has to a particular item in the repository, such as a file or folder.

**artifact**

A Documentum IDE resource, for example an object type, lifecycle, process, or procedure.

**aspect**

An aspect is a BOF module that customizes behavior or records metadata or both for an instance of an object type.

**BOF**

The business object framework (BOF) centralizes business logic within the Documentum framework.

**data dictionary**

The data dictionary stores information about object types and attributes in the repository.

**DAR file**

Documentum ARchive file. A DAR file is the executable, binary version of a Composer project. DAR files are typically used to distribute an application.

**DocApp**

A package that contains the elements that are part of an application. The term DocApp generally refers to applications prior to release 6 that were built using Documentum Application Builder (DAB) instead of Composer.

**DocApp archive**

A pre-version 6 mobile version of a DocApp. The archive file is a compressed (zip) file that can be stored on a local machine or network drive.

**Eclipse**

Eclipse is the name for the overall project supporting the construction of integrated tools for developing applications.

**Eclipse platform**

Eclipse Platform is the name for the core frameworks and services upon which plug-in extensions are created. It provides the runtime in which plug-ins are loaded and run.

**EMF**

Eclipse Modeling Framework. A modeling framework and code generation facility for building tools and other applications based on a structured data model. For more information about EMF, see [Eclipse Modeling Framework](#).

**IDE**

An integrated development environment (IDE), also known as integrated design environment and integrated debugging environment, is a type of computer software that assists computer programmers in developing software.

**lifecycle**

A lifecycle encodes business rules for changes in the properties of an object in a repository, such as a document, as it moves through the stages of its life.

**lightweight object type**

Lightweight objects are part of an object model enhancement introduced to share system managed metadata among objects which only hold application specific data.

**permission set**

Permission sets (also known as ACLs, or access control lists) specify the access each user has to a particular item in the repository, such as a file or folder.

**project**

A project contains the source code and related files for building a program.

**relation type**

A relation type defines the relationship between two objects in a repository.

**repository**

The repository stores the persistent objects managed by Content Server, such as the object metadata and content files.

**SBO**

A service-based object (SBO) is a type of module, like a session bean in an Enterprise JavaBean (EJB) environment. SBOs can operate on multiple object types, retrieve objects unrelated to Documentum objects (for example, external email messages), and perform processing.

**smart container**

Smart containers define objects and relationships in a template that at runtime is used to instantiate instances that are based on the template.

**TBO**

A type-based object (TBO) is an extension of a basic Documentum Foundation Class (DFC) type and can be used to provide a new behavior for existing or new object types, or to customize low-level operations to enforce data validation, referential integrity and business rules.

**Web services**

A Web service (also Web Service) is a software system designed to support interoperable machine-to-machine interaction over a network. Composer supports Web services by providing an EMC Documentum Foundation Services (DFS) registry plug-in.

**workflow**

A workflow formalizes a business process such as an insurance claims process or an engineering development process.

**Workspace**

A workspace is the general umbrella for managing resources in the Eclipse Platform.





## A

### ACL

- basic permissions, 150
- creating, 149
- extended permissions, 150
- overview, 149
- public, 149
- regular, 149
- template, 149, 151

ad hoc relation type, 159

### adding

- alias, 81
- alias set, 81

### alias set

- creating, 81
- details, 83
- editor, 81
- understanding, 81

### Ant task

- building project, 220
- generating DAR, 221
- importing project, 216
- installing DAR file, 221

AntRunner, 15

### artifact

- copying, 31
- creating, 28
- importing, 31
- install options, 203

artifacts, list of available, 28

### aspect

- attaching to type, 179
- constraint expressions, 89
- description, 87
- module, 87
- UI information, 92

aspect editor, 88

aspect module, 141

- configuring deployment, 97
- creating, 95

DFC version, 99

editor, 95

Java system properties, 99

Java VM version, 99

local resources, 100

name, 96

runtime environment, 99

statically deployed classes, 100

type, 96

version requirements, 99

aspect type, creating, 87

### attribute

- constraints, 186
- object type, 184
- SysObject, 173
- value mapping, 190

## B

building, project, 199

## C

### catalog services

- configuring, 67
- editor, 75

### command

- emc.build, 220
- emc.dar, 221
- emc.importProject, 216
- emc.install, 221

### Composer

- architecture, 15
- configuring workspace, 19
- installing, 16

configuration file, XML application, 196

### configuring

- aspect module deployment, 97
- Composer workspace, 19
- connection broker, 18
- Java JRE, 19

- module deployment, 144
- connection broker, configuring, 18
- converting
  - DocApp archive, 46
  - from previous versions, 43
- copying, artifacts, 31
- creating
  - aspect module, 95
  - aspect type, 87
  - JAR definition, 107
  - module definition, 141
  - object type, 176

## D

- DAR file
  - generating, 205
  - generating using Ant, 221
  - installing using Ant, 221
- DAR files
  - installing using the DAR Installer, 209
- DAR Installer Plug-in, 209
- DFS
  - catalog services, 67
  - module options, 65
  - services library, 66
- display configuration, object type, 193
- DocApp
  - importing, 43
- DocApp archive
  - converting, 46
  - importing, 47
- document owner, lifecycle editor, 129
- document renderer, lifecycle editor, 129
- document, smart container, 167
- Documentum artifacts, 28
- Documentum Solutions perspective, 69

## E

- EAR file, 78
- editor
  - alias set, 81
  - aspect module, 95
  - aspect type, 88
  - catalog services, 75
  - format, 103
  - JAR file, 108
  - Java library, 109
  - job, 137

- lifecycle, 112
- lightweight object type, 180
- method, 135
- module definition, 141
- permission set, 154
- relation type, 160
- smart container, 164
- standard object type, 176
- SysObject, 171
- XML, 196
- emc.build command, 220
- emc.dar command, 221
- emc.importProject command, 216
- emc.install command, 221
- enabling tracing, 40
- events, object type, 184
- exporting, Web service, 78

## F

- folder, smart container, 165
- format editor, 103

## G

- generating, DAR file, 221

## H

- headless Composer
  - building project, 220
  - generating DAR, 221
  - installing, 18
  - installing DAR, 221

## I

- importing
  - artifacts, 31
  - DocApp, 43
  - DocApp archive, 47
  - external project, 23
  - Web service, 71
- input mask, 189
- input parameter file, 208
- install
  - artifact options, 203
  - parameters, 200
  - project options, 200
- installing
  - Composer, 16

- DAR file using Ant, 221
  - DAR file using the DAR Installer, 209
  - project using Composer, 200
- ## J
- JAR
    - editor, 108
    - overview, 107
  - JAR definition, creating, 107
  - Java library
    - editor, 109
    - linking, 109
    - overview, 107
  - job editor, 137
  - JRE, configuring, 19
- ## L
- lifecycle
    - document owner, 129
    - document renderer, 129
    - editor, 112
    - introduction, 111
    - location links, 125
    - object types, 112
    - permission sets, 130
    - post-change information, 131
    - properties, 113
    - repeating attributes, 120
    - state actions, 120
    - state attributes, 131
    - state entry criteria, 118
    - state type, 118
    - states, 115
    - version labels, 124
  - lightweight object
    - creating, 179
    - description, 175
    - materialization, 181
  - linking, Java library, 109
  - localization template, 34
  - localizing projects, 34
  - location links, lifecycle editor, 125
- ## M
- materialization, 181
  - method editor, 135
  - module
    - aspect, 141
    - class name, 97, 144
    - configuring deployment, 144
    - creating, 141
    - DFC version, 146
    - editor, 141
    - implementation JARs, 97, 143
    - interface JARs, 97, 144
    - Java system properties, 146
    - Java VM version, 146
    - local resources, 147
    - name, 142
    - runtime environment, 145
    - service-based (SBO), 141
    - statically deployed classes, 146
    - type, 143
    - type-based (TBO), 141
    - version requirements, 146
- ## O
- object
    - smart container, 163
    - SysObject, 171
  - object type
    - attribute, 184
    - attribute constraints, 186
    - attribute structure, 185
    - attribute UI, 188
    - creating, 176
    - definition, 175
    - display configuration, 193
    - events, 184
    - input mask, 189
    - UI, 192
- ## P
- permission set
    - creating, 149
    - lifecycle, 130
    - public, 153
    - regular, 153
    - template, 151
  - permission set editor, 154
  - permissions
    - basic, 150
    - extended, 150
    - overview, 149
  - perspective, Documentum Solutions, 69
  - placeholder, smart container editor, 169

- post-change information, lifecycle
  - editor, 131
- procedure editor, 157
- project, 21
  - building, 199
  - building using Ant, 220
  - checking into source control, 225
  - configuring DFS module, 65
  - creating, 22
  - creating from DocApp, 43
  - creating from DocApp archive, 47
  - DFS services library, 66
  - import from source control system, 226
  - importing, 23
  - input parameter file, 208
  - install options, 200
  - installation procedures, 202
  - localizing, 34
  - new, 22
  - referencing, 27
- properties
  - alias set, 83
  - lifecycle, 113
- publish, Web service, 76

## R

- referencing, other projects, 27
- relation type
  - ad hoc, 159
  - editor, 160
  - system, 159
- relationship, smart container editor, 170
- repeating attributes, lifecycle editor, 120
- repository, dfc.properties file, 18
- runtime environment
  - aspect module, 99
  - module, 145

## S

- SBO, 141
- service catalog, import service, 71
- service-based module, 141
- smart container
  - definition, 163
  - document, 167
  - editor, 164
  - folder, 165
  - new document, 168

- new folder, 166
- placeholder, 169
- relationships, 170
- template, 168
- source control system
  - building project, 227
  - checking in projects, 225
  - using, 225
- starting, Workflow Manager, 17
- state
  - actions, 120
  - attributes, 131
  - entry criteria, 118
  - type, 118
- SysObject
  - attributes, 173
  - definition, 171
  - editor, 171
- system relation type, 159

## T

- TBO, 141
- template, smart container, 168
- tracing, enabling, 40
- type
  - attaching aspects, 179
  - attribute, 184
  - attribute input mask, 189
  - constraint expressions, 183
  - editor, 180
- type editor, 176
- type UI, 192
- type-based module, 141

## V

- value mapping, object type, 190
- version labels, lifecycle editor, 124

## W

- Web services
  - catalog services, 67
  - consuming a service, 72
  - DFS module, 65
  - Documentum Solutions
    - perspective, 69
  - exporting, 78
  - importing a service, 71
  - publishing, 76

WSDL, 75  
Workflow Manager, 17  
WSDL, 75

configuration file, 196  
description, 195  
editor, 196

## **X**

XML application